HC(S)08/RS08
Debugger Manual

Revised: 5 May 2006 - HC08DEBUGGERRM REV 1

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. CodeWarrior is a trademark or registered trademark of Freescale Semiconductor, Inc. in the United States and/or other countries. All other product or service names are the property of their respective owners.

Copyright © 1989 - 2006 by Freescale Semiconductor, Inc. All rights reserved.

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

How to Contact Us

Corporate Headquarters	Freescale Semiconductor, Inc. 7700 West Parmer Lane Austin, TX 78729 U.S.A.
World Wide Web	http://www.freescale.com/codewarrior
Technical Support	http://www.freescale.com/support

Table of Contents

Introduction

Manual Contents 1

Book I - Debugger Engine

Book I Contents 3

1 Introduction 5

Freescale Debugger. 5
Debugger Application. 5
Debugger Features 6
Demo Version Limitations on Components 6

2 Debugger Interface 7

Introduction. 7
Application Programs 7
Starting the Debugger 8
Starting from within the IDE. 8
Starting Debug from the Project Window 8
Starting Debug from the Main Window Menu Bar 8
Debugger Command Line Start. 9
Command Line Options. 9
Order of Commands 11
Debugger Main Window. 12
Debugger Main Window Toolbar 12
Debugger Main Window Status Bar 13

Main Window Menu Bar 13
File Menu. 14
Preferences Window 16
View Menu. 19
Customizing the Toolbar 19
Run Menu 22
Connection Menu 24
Loading a Connection 25
Connection Command File Window 27
Component Menu 29
Window Menu. 31
Help Menu. 33
About Box 33
Component Associated Menus 34
Component Main Menu 34
Component Files 35
Component Windows Object Info Bar 35
Component Popup Menu. 35
Highlights of the User Interface 36
Activating Services with Drag and Drop 36
To Drag and Drop an Object 38
Drag and Drop Combinations 39
Dragging from Assembly Component Window 39
Dragging from Data Component Window. 39
Dragging from Source Component Window 40
Dragging from the Memory Component Window 41
Dragging from Procedure Component Window 42
Dragging from Register Component Window. 42
Dragging from Module Component Window 43
Selection Dialog Box. 43

3 Debugger Components 45

Component Introduction 45
CPU Components 45
Window Components 45

Connection Components 46
Loading Component Windows 46
General Debugger Components 48
Assembly Component 48
Assembly Menu 49
Setting Breakpoints 49
Associated Popup Menu 50
Command Line Component 53
Command Menu 55
Cache Size 56
Coverage Component 57
Coverage Operations 58
Coverage Menu 58
Output File 59
Split View Associated Popup Menu 60
DA-C Link Component 62
DA-C Link Operation 62
DA-C Link Menu 62
Drag Out 63
Drop Into 63
Demo Version Limitations 63
Data Component 64
Data Operations 64
Expression Editor 65
Expression Command file 66
Data Menu 67
Scope Submenu 67
Format Submenu 68
Format Selected & All Sub Menu 69
Mode Submenu 69
Options Submenu 71
Zoom and Sort Submenus 72
Associated Popup Menu 72
Memory Component 76
Memory Operations 77

Memory Menu 77
Display Submenu 80
Fill Memory 81
Display Address 81
CopyMem Submenu 83
Search Pattern 83
Update Rate 84
Associated Popup Menu 85
MicroC Component 88
MicroC Link Menu 88
MicroC DLLs 89
Module Component 92
Module Operations 92
Module Menu 92
Procedure Component 94
Procedure Operations 94
Procedure Menu 94
Profiler Component 96
Profiler Operations 97
Profiler Output File Functions 98
Recorder Component 100
Recorder Operations 100
Recorder Menu 101
Register Component 103
Status Register Bits 103
Editing Registers 103
Register Menu (Format Submenu) 104
Drop Into: 105
SoftTrace Component 106
SoftTrace Operations 106
SoftTrace Menu 106
Associated Popup Menu 107
Source Component 109
Folding and Unfolding 112
Source Menus 113

Open Source File 117
Go to Line 117
Find Operations 118
Find Procedure 119
Folding Menu 119
Visualization Utilities 121
Inspector Component 122
Inspector Operations 126
Inspector Menu 127
Associated Popup Menu 127
VisualizationTool Component 129
Edit Mode and Display Mode 130
VisualizationTool Menu 131
Associated Popup Menu 132
VisualizationTool Properties 134
Instruments 134

4 Control Points 145

Introduction 145
Breakpoints 146
Breakpoints Tab 148
Multiple Selections in List Box 149
Checking Expressions 149
Saving Breakpoints 150
Setting Breakpoints 152
Positions Where a Breakpoint Is Definable 152
Temporary Breakpoints 153
Setting Temporary Breakpoints 153
Permanent Breakpoints 154
Setting Permanent Breakpoints 154
Counting Breakpoints 155
Setting Counting Breakpoints 155
Conditional Breakpoints 156
Setting Conditional Breakpoints 156
Deleting Breakpoints 157

Associate a Command with a Breakpoint 158
Demo Version Limitations 158
Watchpoints 159
Watchpoints Tab 161
Multiple Selections 162
Checking Syntax 162
Setting Watchpoints 163
Setting a Read Watchpoint 163
Setting a Write Watchpoint 164
Defining a Read/Write Watchpoint 164
Defining a Counting Watchpoint 165
Defining a Conditional Watchpoint 166
Deleting a Watchpoint 167
Associate a Command with a Watchpoint 167
Demo Version Limitations 168
Markpoints 169
Markpoints Tab 171
Setting Markpoints 172
Setting a Source Markpoint 172
Setting a Data Markpoint 172
Setting a Memory Markpoint 173
Deleting a Markpoint 173
Halting on a Control Point 174

5 Real Time Kernel Awareness 175

Introduction 175
Inspecting Task State 176
RTK Interface 176
Task Description Language 176
Application Example 178
Inspecting Kernel Data Structures 179
OSEK Kernel Awareness 180
OSEK ORTI 180
ORTI File and Filename 181
ORTI File Structure 182

OSEK RTK Inspector Component 182
Inspector Task 184
Inspector Stack. 185
Inspector SystemTimer 185
Inspector Alarm 186
Inspector Message 187

6 How To ... 189

How To Configure the Debugger 190
For Use from Desktop (Win 95, Win 98,Win NT4.0 or Win2000). 190
Defining the Default Directory in the MCUTOOLS.INI 190
How To Start the Debugger 191
From WinEdit 191
Automating Debugger Startup 192
How To Load an Application 193
How To Start an Application 194
How To Stop an Application. 194
How To Step in the Application 195
On Source Level 195
On the Next Source Instruction 195
Step Over a Function Call (Flat Step) 196
Step Out from a Function Call. 196
Step on Assembly Level 197
How To Work on Variables. 197
Display Local Variable from a Function 197
Display Global Variable from a Module 197
Change Format for Variable Value Display 198
Modify a Variable Value 199
Get the Address Where a Variable is Allocated 200
Inspect Memory Starting at a Variable Location Address 200
Load an Address Register with the Address of a Variable 200
How To Work on the Register. 201
Change Format of Register Display 201
Modify a Register Content 201
Modify Index or Accumulator Register Content. 201

Modify Bit Register Content 202
Start Memory Dump at Address Where Register Is Pointing 202
Modify Content of Memory Address 203
How to Consult Assembler Instructions Generated by a Source Statement . . . 203
How To View Code 204
How to Communicate with the Application 205
About startup.cmd, reset.cmd, preload.cmd, postload.cmd 205

7 CodeWarrior Integration 207

Debugger Configuration 207

8 Debugger DDE Capabilities 209

Introduction 209
DDE Implementation 209
Driving Debugger through DDE 209

9 Synchronized Debugging Through DA-C IDE 211

Configuring DA-C IDE for Freescale Tool Kit 211
Create New Project 212
Configure Working Directories 212
Configure File Types 214
Configure Library Path 214
Adding Files to Project 216
Building The Database 217
Configuring The Tools 219
Compiler Configuration 219
Linker Configuration 220
Maker Configuration 221
Debugger Interface 222
DA-C IDE and Debugger Communication 223
Communication DLL Installation 223
Debugger Properties Configuration 224
Debugger Project File Configuration 225
Synchronized Debugging 227
Troubleshooting 227

Book II - HC08 Debug Connections

Book II Contents 231

10 HC08/RS08 Debugging First Steps 233

Technical Considerations 233
Full Chip Simulation Considerations 233
HC08 Serial Monitor Considerations 234
ICS MON08 Interface Connection 234
P&E Multilink/Cyclone Pro Considerations 234
ICS P&E Multilink/Cyclone Pro Considerations 234
SofTec HC08 Considerations 234
Debugging First Steps Using the Wizard 236
Switching Connections 243
Loading the Full Chip Simulation Connection 243
Loading the P&E Multilink/Cyclone Pro Connection 245

11 HC08 Full Chip Simulation 249

Configuration Procedure 249
Select Device Option 250
Full Chip Simulation Module Commands 251
Run Till Cycle Option 251
Initialize Input Ports On Startup 252
View Register Files Option 252
Peripheral Modules Options 253
ADC Module Option 253
ADC User Commands 255
Clock Generation Module Option 256
CGM Commands 258
High Resolution PWM Module 258
High Resolution PWM Commands 260
Input/Output (I/O) Ports Module 262
Input/Output Ports Commands 263
External Interrupt Module 264

IRQ Commands 265
Keyboard Interrupt Module 266
Keyboard Interrupt Commands 267
Multi Master Inter-Integrated Circuit Module 268
Multi-Master Inter-Integrated Circuit Module Commands 270
FCSMSCAN Controller Module 273
FCSMSCAN08 Background and Assembly Example 273
FCSDemo CAN Program 273
FCSMSCAN Commands 286
FCSProgrammable Timer Interrupt Module 287
FCSSerial Communications Interface Module 288
FCSSCI Commands 289
FCSSlave LIN Interface Controller Module 291
FCSSLIC Commands 294
FCSSerial Peripheral Interface Module 296
FCSSPI Commands 298
FCSTimer Interface Module 300
FCSTimer Interface Module Commands 303
FCSUniversal Serial Bus (USB) Module 305
USB Commands 324

12 MON08 Interface Connection 329

Connection Procedure 329
Advanced Settings Tab 331
Tpd And Tpu Timing Listboxes 331
Target Has RESET Button 332
MON08 Cable Connection Communications Type 332
Power Up and Power Down Radio Buttons 332
Turn Target Power Off and Leave Target Power On Radio Buttons . . . 333
Serial Port Stop Bits 334
Pulse IRQ on Stop 334
Target MCU Security Bytes 335
IGNORE Security Failure And Enter Monitor Mode 335
STATUS Area 336
Active Mode Connection Menu Options 339

Advanced Programming/Debug Options Option 339
View Register Files Option 341
Device Class Description 343
Class 1 Device 343
Class 2 Device 343
Class 3 Device 343
Class 4 Device 344

13 ICS MON08 Interface Connection 345

ICS Mode 345
Connection Procedure 345
Advanced Settings Tab 348
Tpd And Tpu Timing Textboxes 348
Target Has RESET Button 349
MON08 Cable Connection Communications Type 349
Power Up and Power Down Radio Buttons 349
Cycle Power Radio Button 350
Serial Port Stop Bits 351
Pulse IRQ on Stop 351
Target MCU Security Bytes 351
IGNORE Security Failure And Enter Monitor Mode 351
STATUS Area 352
Active Mode Connection Menu Options 354
Advanced Programming/Debug Options Option 354
Start Expert Mode Programmer Option 355
View Register Files Option 356
Device Class Description 358
Class 1 Device 358
Class 2 Device 358
Class 3 Device 358
Class 4 Device 359

14 P&E Multilink/Cyclone PRO Connection 361

Connection Procedure 361
Advanced Settings Tab 365

Tpd And Tpu Timing Textboxes 365
Target Has RESET Button Checkbox 366
MON08 Cable Connection Communications Type 366
Power Up and Power Down Radio Buttons 366
Cycle Power Down/Up Radio Button 367
Serial Port Stop Bits 368
Pulse IRQ on Stop 368
Target MCU Security Bytes 368
IGNORE Security Failure And Enter Monitor Mode 369
STATUS Area 370
Connection (MultilinkCyclonePro) Menu 372
Device Option 372
Connect Option 373
Active Mode Menu Options 374
Advanced Programming/Debug Options Option 374
Start Expert Mode Programmer Option 375
View Register Files Option 376
Debugging Limitations 377
Debugging Tips 378
Device Class Description 379
Class 5 Device 379
Class 7 Device 379
Class 8 Device 379

15 ICS P&E Multilink/Cyclone PRO Connections 381

Connection Procedure 382
Advanced Settings Tab 385
Tpd And Tpu Timing Textboxes 385
Target Has RESET Button 386
MON08 Cable Connection Communications Type 386
Power Up/Down Radio Buttons 386
Cycle Power Up/Down Radio Buttons 387
Target MCU Security Bytes 388
STATUS Area 388
Active Mode Menu Options 391

Advanced Programming/Debug Options Option 391
View Register Files Option 393
Device Class Description 394
Class 5 Device 394
Class 7 Device 394
Class 8 Device 394

16 Softec HC08 Connection 395

SofTec HC08 Technical Considerations 395
CodeWarrior and SofTec HC08 Connection. 395
Using the Stationery Wizard 396
From Within an Existing Project 397
inDart-HC08 Menu Options 398
MCU Configuration Option. 399
User's Manual Option 399
About Option 399
MCU Configuration Dialog Box. 399
Communication Settings Dialog Box 400
About Dialog Box 401

17 FSICEBASE Emulator 403

FSICEBASE Overview. 403
System Requirements 403
System Features. 404
System Components 405
Basic Components 405
Additional Components 406
Setting Up The FSICEBASE System 406
Setting Up The Hardware 407
Establishing Communication 408
Communication Through Ethernet Port 408
Communication Through USB Port 410
Setting Up The System 410
Specifying A Connection 411
Specifying Communication Information. 412

Assigning An IP Address To FSICEBASE. 413
Specifying a Memory Map 416
Specifying the Clock Speed 419
Emulation System Reset 421
Setting Up Logic Cables And Connectors 422
Bus State Analyzer (BSA) 423
Using BSA. 423
Defining Events 424
Recording Modes. 426
Time Tag Clock Frequency 428
Collecting Bus Data. 428
Viewing Data 429

18 MMDS0508 Emulator 431

MMDS General Description. 431
MMDS Features 431
MMDS Components 432
MMDS Installation 433
Configuring the Platform Board 435
Factory Test Header (J1) 435
Port Voltage Control Headers (J2–J4). 435
Installing the EM 436
Removing the EM. 437
Making Cable Connections. 437
Host Computer Connection. 437
Bus State Analyzer Connection. 437
Target Cable Connection 438
Power Connection 439
Connector Information 439
Reset Switch 439
RS-232 Serial Connector. 439
Logic Cables and Connectors 440
Power Supply Fuse Replacement 442
MMDS Connection. 443
MMDS0508. 444

Configuration 444
Interfacing Your System and the Connection 444
Hardware Connection 444
Loading the MMDS0508 Connection. 445
Communication Configuration 446
Communication Device Specification. 447
Data Format. 448
MMDS Debugger Status Bar 448
MMDS0508 Menu Options 448
Communication Baud Rate 449
Communication Option 449
Maximum Baud Rate. 450
Show Protocol 450
Memory Map Option. 450
Personality (.MEM) Files 451
Dual-Port RAM 453
Memory 453
Signals Emulation Option 453
MCU Clock 454
Reset 454
Bus Tracing 454
Default Connection Setup. 455
Freescale ESL Parameters. 455
COMDEV 455
BAUDRATE 456
SHOWPROT 456
Bus Analyzer. 456
Watchpoints 456
Trace Modes 456
Trace Buffer. 457
Textual or Graphic Display 457
Using the Bus Analyzer. 457
Trace Group Popup Menu. 458
Trigger Setup 458
Sequencer Setup 460

Non-Triggered Modes 461
Counted Modes 461
Triggered Modes/Sequential Event Mode 462
Counted/Sequential Recording Mode 463
Time Tag Clock Setup 463
Collecting Data 464
Arming the Analyzer 465
Disarming the Analyzer 465
Start Emulation 465
Status Bar 465
Halt Data Collection 465
Halt Emulation 465
Recording Bus Data 465
Trigger Event 466
Viewing Collected Data 466
View Cycles 466
Textual, Graphical or Instructions 466
Textual Display 466
Instructions Only Display 468
Graphical Display 468
Scrolling the Display 471
Search for Frame 472
Search for Events 472
Search for a Pattern 474
Commands in the HI-WAVE Run Menu 475
Continue 475
Halt 476
Step 476
Flat 476
Asm Step 476
Dumping Bus Analyzer Data to File 476
Dump File 477

19 MMEVS0508 Emulator 481

MMEVS General Description 481

MMEVS Features 481
MMEVS Components 482
MMEVS Installation 482
Configuring the Platform Board 483
Factory Test Header (J1) 483
Port Voltage Control Headers (J2–J4) 484
Installing the EM 485
Removing the EM 485
Making Cable Connections 485
Host Computer Connection 485
Target Cable Connection 486
Power Connection 486
Reset Switch 487
RS-232 Serial Connector 487
MMEVS Connection 488
MMEVS0508 488
MMEVS0508 Connection Menu 489
Interfacing Your System and Connection 489
Hardware Connection 489
Loading the MMEVS0508 Connection 490
Communication Configuration 491
Communication Device Specification 492
Data Format 493
MMEVS Debugger Status Bar 493
MMEVS0508 Menu Options 493
Communication Baud Rate 494
Communication 494
Maximum Baud Rate 494
Show Protocol 494
Memory Configuration 495
Personality (.MEM) Files 495
Memory 498
Signals Emulation 498
MCU Clock 498
Reset 499

Default Connection Setup 499
Freescale ESL Parameters 499
COMDEV 500
BAUDRATE 500
SHOWPROT 500
. 500

Book III - HCS08 Debug Connections

Book III Contents 501

20 HCS08 Full Chip Simulation 503

Configuration Procedure 503
Connection (HCS08FCS) Menu 504
Device Option 504
Full Chip Simulation Module Commands 505
Peripheral Modules Commands 507
ADC Module 507
ADC Module Commands 509
Clock Generation Module 509
Clock Generation Module Commands 511
Inter-Integrated Circuit Module 512
Inter-Integrated Circuit Module Commands 513
Input/Output (I/O) Ports Module 516
Input/Output Ports User Commands 517
External Interrupt (IRQ) Module 518
IRQ Commands 519
Keyboard Interrupt Module 520
Keyboard Interrupt Commands 522
Modulo Timer Interrupt Module 523
Serial Communications Interface Module 524
SCI Commands 525
Serial Peripheral Interface Module 527

SPI Commands 529
Timer Interface Module. 532
Timer Module Commands 534

21 HCS08 P&E Multilink/Cyclone PRO Connection 537

Connection Procedure 537
Hotsync Button 539
MultilinkCyclonePro Menu Description 540
Device Option 540
Connect Option 542
Active Mode Menu Options 542
Advanced Programming/Debug Options Option 542
View Register Files Option 544
Trigger Module Settings Option 545
Bus Trace Option 545

22 Softec HCS08 Connection 547

SofTec HCS08 Technical Considerations 547
CodeWarrior and SofTec HCS08 Connection 547
First Steps Using the Stationery Wizard 548
First Steps From Within an Existing Project 550
SofTec HCS08 Menu Options 551
MCU Configuration Option. 552
About Option 552
Trigger Module Settings 552
Bus Trace 552
MCU Configuration Dialog Box. 552
Communication Settings Dialog Box 553
About Dialog Box 554

23 HCS08 Serial Monitor Connection 555

Serial Monitor Technical Considerations 555
CodeWarrior and Serial Monitor Connection 555
First Steps Using the Stationery Wizard 556
First Steps From Within an Existing Project 558

MONITOR-HCS08 Menu Options . . . 562
Monitor Communication... . . . 562
Vector Mirroring Setup... . . . 562
Erase Flash . . . 562
Trigger Module Settings... . . . 562
Bus Trace . . . 563
Select Derivative . . . 563
Monitor Setup Window . . . 563
Monitor Communication Tab . . . 563
Vector Table Mirroring Tab . . . 564
Derivative Selection Dialog Box . . . 565

24 HCS08 On-chip DBG Module 567

Reference Document . . . 567
DBG Features . . . 567
Specific Connection Menu Options . . . 568
Popup Menu Entries in Source, Data, Assembly and Memory Windows . . . 568
Source and Assembly Windows . . . 568
Trigger Storing as Markpoints . . . 571
Data and Memory Windows . . . 574
Expert Triggers . . . 575
Trigger Settings . . . 578
Trigger Module Usage . . . 579
DBG Support Status Bar Item . . . 579
Trigger Module Settings Window . . . 580
DBG Module Mode Setup . . . 581
Automatic Mode (Default) . . . 581
Expert Mode . . . 582
Expert Mode Tab . . . 583
Profiling and Coverage Mode . . . 583
Disabled Mode . . . 584
“Memory Access” Triggers . . . 585
Memory Access at Address A . . . 585
Memory Access at Address A or Address B . . . 585
Memory Access Inside Address A - Address B Range . . . 585

Memory Access at Address A then Memory Access at Address B. . . . 585
Memory Access at Address A and Value on Data Bus Match 585
Memory Access at Address A and Value on Data Bus Mismatch. 586
“Instruction” Triggers 587
Instruction at Address A Is Executed 587
Instruction at Address A or Address B Is Executed 587
Instruction Execution Inside Address A - Address B Range". 588
Instruction Execution Outside Address A - Address B Range" 588
Instruction at Address A then at Address B Were Executed 588
Instruction at Address A and Value on Data Bus Match 588
Instruction at Address A and Value on Data Bus Mismatch 589
“Capture” Triggers 590
Capture Read/Write Values at Address B 590
Capture Read/Write Values at Address B After Access at Address A . 590
DBG Module Options 591
Program Code Change of Flow Recording 591
When Data Recording 592
Trigger Editing 593
Trigger Module Settings Window - Display Information 595
General Settings Tab 596
Trace Component Window 597
Instructions Display 598
Graphical Display 599
Textual Display 599
Column Display and Moving 600
Dumping Frames to File 601
Goto Frame 601
Clearing Frames. 602
DBG Module FIFO/Buffer Display. 602
Recorded Data Display 603
Limitations 604

Book IV - RS08 Debugger Connections

Book IV Contents 605

25 RS08 Full Chip Simulation 607

Configuration Procedure 607
Connection (RS08FCS) Menu 608
Device Option 608
Full Chip Simulation Module Commands 609
Peripheral Modules Commands 612

26 RS08 P&E Multilink/Cyclone PRO Connection 613

Connection Procedure 613
Hotsync Button 615
MultilinkCyclonePro Menu Description 616
Device Option 616
Connect Option 617
Active Mode Menu Options 617
Advanced Programming/Debug Options 617
Show Processor Pins 619
View Register Files Option 620

27 SofTec RS08 Connection 623

SofTec RS08 Technical Considerations 623
CodeWarrior and SofTec RS08 Connection 623
First Steps Using the Stationery Wizard 624
First Steps From Within an Existing Project 626
SofTec RS08 Menu Options 627
MCU Configuration Option 628
About Option 628
MCU Configuration Dialog Box 628
Communication Settings Dialog Box 628
About Dialog Box 630

Book V - Commands

Book V Contents 631

28 Debugger Engine Commands 633

Commands Overview 633
Command Syntax 634
Available Command Lists 634
Kernel Commands 634
Base Commands 635
Environment Commands 637
Component Commands 638
Component Specific Commands 639
Command Syntax Terms 642
Module Names 643
Debugger Commands 643
A 644
ACTIVATE 644
ADDXPR 645
ATTRIBUTES 645
In the Command Component 645
In the Procedure Component 646
In the Assembly Component 646
In the Register Component 647
In the Source Component 649
In the Data Component 650
In the Memory Component 651
In the Inspector Component 653
AT 655
AUTOSIZE 655
BASE 656
BC 656
BCKCOLOR 657
BD 658

BS 659
CALL 661
CD 662
CF 663
CLOCK 665
CLOSE 665
COPYMEM 666
CMDFILE 666
CR 667
CYCLE 667
DASM 668
DB 669
DDEPROTOCOL 671
DEFINE 672
DETAILS 673
DL 674
DUMP 675
DW 675
E 676
ELSE 677
ELSEIF 677
ENDFOCUS 678
ENDFOR 679
ENDIF 679
ENDWHILE 680
EXECUTE 680
EXIT 681
FILL 681
FILTER 682
FIND 683
FINDPROC 684
FOCUS 685
FOLD 686
FONT 686
FOR 687

FPRINTF 688
FRAMES 688
G 689
GO 690
GOTO 691
GOTOIF 692
GRAPHICS 693
HELP 693
IF 694
INSPECTOROUTPUT 695
INSPECTORUPDATE 695
LF 696
LOAD 697
LOADCODE 698
LOADSYMBOLS 698
LOG 699
More About Logging of IF, FOR, WHILE and REPEAT 700
LS 703
MEM 704
MS 705
NB 706
NOCR 708
NOLF 708
OPEN 709
OUTPUT 710
P 711
PAUSETEST 712
PRINTF 713
PTRARRAY 713
RD 714
RECORD 715
REPEAT 715
RESET 716
RESTART 716
RETURN 717

RS 718
S. 719
SAVE. 720
SAVEBP 721
SET 722
SETCOLORS 722
SLAY. 723
SLINE 723
SMEM. 724
SMOD 725
SPC 726
SPROC 727
SREC. 728
STEPINTO 729
STEPOUT 730
STEPOVER. 731
STOP 732
T 733
TESTBOX. 734
TUPDATE. 734
UNDEF 735
UNFOLD. 737
UNTIL. 738
UPDATERATE 738
VER. 739
WAIT. 740
WB 741
WHILE 742
WL. 743
WW 743
ZOOM. 744

29 Debugger Connection-specific Commands 745

MMDS Commands 745
Baud Rate Commands. 745

BAUD: Set Communication Baud Rate 745
Trigger Commands 746
CT: Clear Triggers 746
ST: Set Trigger 747
TD: Trigger Disable 750
TE: Trigger Enable 751
Bus Analyzer Commands 752
ARM: Arm Bus Analyser 752
DARM: Disarm Bus Analyser 752
GE: Go To Event 753
GF: Go To Frame 753
GP: Go to Analyzer Search Pattern 754
LT: Log Trace 755
SC: Set Analyzer Clock 756
SP: Set Analyzer Search Pattern 757
SQ: Set Sequencer 760
TT: Display Time Tag Difference 761
VA: Analyzer View 762
Target Signal Emulation Commands 763
SIG: Set Emulator Signals 763
Reset Commands 764
RESET: Reset Target MCU 764
Other Commands 765
LOADMAP: Load Memory Map 765
MEM: Display Memory Map 766
OSC: Select Emulator Clock Frequency 767
PROTOCOL: Controls Show Protocol Functionality 767
RTMEM: Configure Real Time Memory 768
MMEVS Commands 769
Baud Rate Commands 769
BAUD: Set Communication Baud Rate 769
Target Signal Emulation Commands 770
SIG; Set Emulator Signals 770
Reset Commands 771
RESET: Reset Target MCU 771

Other Commands. 772
LOADMAP: Load Memory Map 772
MEM: Display Memory Map 773
OSC: Select Emulator Clock Frequency 774
PROTOCOL: Controls Show Protocol Functionality 774

Book VI - Environment Variables

Book VI Contents 775

30 Debugger Engine Environment Variables 777

Debugger Environment. 778
The Current Directory 779
Global Initialization File (MCUTOOLS.INI - PC Only) 780
Local Configuration File (usually project.ini) 781
Default Layout Configuration (PROJECT.INI) 782
Ini File Activation 784
Environment Variable Paths 785
Line Continuation 786
Environment Variables 787
ABSPATH: Absolute Path. 788
DEFAULTDIR: Default Current Directory. 789
ENVIRONMENT=: Environment File Specification. 790
GENPATH: #include “File” Path 791
LIBRARYPATH: ‘include <File>’ Path 792
OBJPATH: Object File Path 793
TMP: Temporary directory 794
USELIBPATH: Using LIBPATH Environment Variable 795
Search Order for Source Files. 796
In the Debugger for C Source Files (*.c, *.cpp) 796
In the Debugger for Assembly Source Files (*.dbg). 796
In the Debugger for Object Files (HILOADER). 796
Debugger Files 796

31 Connection-specific Environment Variables 799

Connection-specific Environment Variables 799

Book VII - Appendixes

Book VII Contents 801

A Debugger Legacy 803

Legacy PEDebug Target Interface 803

New P&E Connections for HC08/HCS08 803

“Revert to Full Chip Simulator” Feature Removed 803

Connection Selection or Change Always Available within IDE. 804

Automatic Upgrade Path for Projects Previously Created 804

Legacy Target Interfaces Removed. 804

HC08 and HCS08 Full Chip Simulator Components NO LONGER Supported . . 804

Introduction

Manual Contents

The HC(S)08/RS08 Debugger Manual consists of the following books:

Book 1: Debugger engine - defines the HC08 and HC(S)08 common and base features, their functionality, and a description of the components that are avilable in the debugger.

- "Introduction" on page 5
- "Debugger Interface" on page 7
- "Debugger Components" on page 45
- "Control Points" on page 145
- "Real Time Kernel Awareness" on page 175
- "How To ..." on page 189
- "CodeWarrior Integration,"
- "Debugger DDE Capabilities,"
- "Synchronized Debugging Through DA-C IDE,"

Book 2: HC08 Debugger Connections - defines the connections available for debugging code written for HC08 CPUs.

- "HC08 Full Chip Simulation" on page 249
- "MON08 Interface Connection" on page 329
- "ICS MON08 Interface Connection" on page 345
- "P&E Multilink/Cyclone PRO Connection" on page 361
- "ICS P&E Multilink/Cyclone PRO Connections" on page 381
- "Softec HC08 Connection" on page 395
- "FSICEBASE Emulator" on page 403
- "MMDS0508 Emulator" on page 431
- "MMEVS0508 Emulator" on page 481

Book 3: HCS08 Debugger Connections - defines the connections available for debugging code written for HCS08 CPUs

- "HCS08 Full Chip Simulation" on page 503
- "HCS08 P&E Multilink/Cyclone PRO Connection" on page 537
- "Softec HCS08 Connection" on page 547
- "HCS08 Serial Monitor Connection" on page 555

Book 4: Debugger Connections common features

- "HCS08 On-chip DBG Module" on page 567

Book 5: Commands

- Debugger Engine Commands on page 633
- Debugger Connection-specific Commands on page 745

Book 6: Environment Variables

- Debugger Engine Environment Variables on page 777
- Connection-specific Environment Variables on page 799

This manual also contains an Appendix that discusses the Debugger Legacy

- Appendix A: "Debugger Legacy" on page 803

Book I - Debugger Engine

Book I Contents

Each section of the Debugger manual includes information to help you become more familiar with the Debugger, to use all its functions and help you understand how to use the environment. This book, the Debugger engine, defines the HC08 and HC(S)08 common and base features, their functionality, and gives a description of the components that are available in the debugger.

This book is divided into the following chapters:

- This chapter describes the manual and special features of the Debugger.
- The "Introduction" on page 5 Chapter introduces the Debugger concept.
- The "Debugger Interface" on page 7 Chapter provides all details about the Debugger user interface environment i.e., menus, toolbars, status bars and drag and drop facilities.
- The "Debugger Components" on page 45 Chapter contains descriptions of each basic component and visualization utility.
- The "Control Points" on page 145 Chapter is dedicated to the control points and associated windows.
- The "Real Time Kernel Awareness" on page 175 Chapter contains descriptions of the Real Time concept and related applications.
- The "How To ..." on page 189 Chapter provides answers for common questions and describes how to use advanced features of the Debugger.
- The "CodeWarrior Integration," chapter explains how to configure the Debugger for use with CodeWarrior.
- The Debugger DDE Capabilities on page 209 describe the debugger DDE features.
- The Synchronized Debugging Through DA-C IDE on page 211 chapter explains the use of tools with the DA-C IDE from RistanCase

1

Introduction

This section is an introduction to the Debugger from Freescale used in 8/16 bit embedded applications.

Freescale Debugger

The Debugger is a member of the tool family for Embedded Development. It is a Multipurpose Tool that you can use for various tasks in the embedded system and industrial control world. Some typical tasks are:

- Simulation and debugging of an embedded application.
- Simulation and debugging of real time embedded applications.
- Simulation and/or cross-debugging of an embedded application.
- Multi-Language Debugging: Assembly, C and C++
- True Time Stimulation
- User Components creation with the Peripheral Builder
- Simulation of a hardware design (e.g., board, processor, I/O chip).
- Building a target application using an object oriented approach.
- Building a host application controlling a plant using an object oriented approach.

Debugger Application

A Debugger Application contains the Debugger Engine and a set of debugger components bound to the task that they should perform (for example a simulation and debugging session). The Debugger Engine is the heart of the system. It monitors and coordinates the tasks of the components. Each Debugger Component has its own functionality (e.g., source level debugging, profiling, I/O stimulation).

You can adapt your Debugger application to your specific needs. Integrating or removing the Debugger Components is very easy. You can add additional Debugger Components (for example, for simulation of a specific I/O peripheral chip) and integrate them with your Debugger Application.

You can also open several components of the same type.

Debugger Features

- True 32-bit application
- Powerful features for embedded debugging
- Special features for real time embedded debugging
- Powerful features for True Time Simulation
- Various and Same look Target Interfaces
- User Interface
- Versatile and intuitive drag and drop functions between components
- Folding and unfolding of objects like functions, structures, classes
- Graphical editing of user defined objects
- Visualization functions
- Smart interactions with objects
- Extensibility function
- Both Powerful Simulation & Debugger
- Show Me How Tool
- GUI (graphical user interface) version including command line
- Context sensitive help
- Configurable GUI with Tool Bar
- Smooth integration into third party tools
- Supports both Freescale and ELF/DWARF Object File Format and S-Records.

Demo Version Limitations on Components

When the Debugger is started in demo mode or with an invalid engine license, then all components that are protected with FLEXlm are in demo mode. The limitations of all components are described in their respective chapter.

2

Debugger Interface

This chapter describes the Debugger Graphic User Interface (GUI). Click any of the following links to jump to the corresponding section of this chapter:

- Introduction on page 7
- Application Programs on page 7
- Starting the Debugger on page 8
- Debugger Main Window on page 12
- Component Associated Menus on page 34
- Highlights of the User Interface on page 36

Introduction

The CodeWarrior IDE main window acts as a container for windows of all the debugger components. The main window provides a main menu bar, a tool bar, a status bar for status information, and object information bars for several components.

The Debugger main window allows you to manage the layout of the different component windows (**Window** menu of the Debugger application). Component windows are organized as follows:

- Tiled arrangement - Auto tiled, component windows are automatically resized when the main window is resized
- Component windows are overlapped
- Component windows that are currently minimized are Debugger Main windoe icons.

Application Programs

The CodeWarrior installer places executable programs in the `prog` subdirectory of the CodeWarrior installation directory. For example, if you installed the CodeWarrior IDE software in `C:\Freescale`, you would find all program files in the folder `C:\Freescale\prog`.

The following list is an overview of files that CodeWarrior uses for C/C++ debugging.

`hiwave.exe` Debugger executable file

`hibase.dll` Debugger main function dll

`elfload.dll` Debugger loader dll

`*.wnd` Debugger component

`*.tgt` Debugger target file

`*.cpu` Debugger CPU awareness file

Starting the Debugger

This section explains how you can start the debugger from within the Codewarrior IDE or from a DOS command line.

Starting from within the IDE

There are two ways to start the debugger from within the IDE, from a **Project** window icon, or from the IDE Main Window menu bar.

Starting Debug from the Project Window

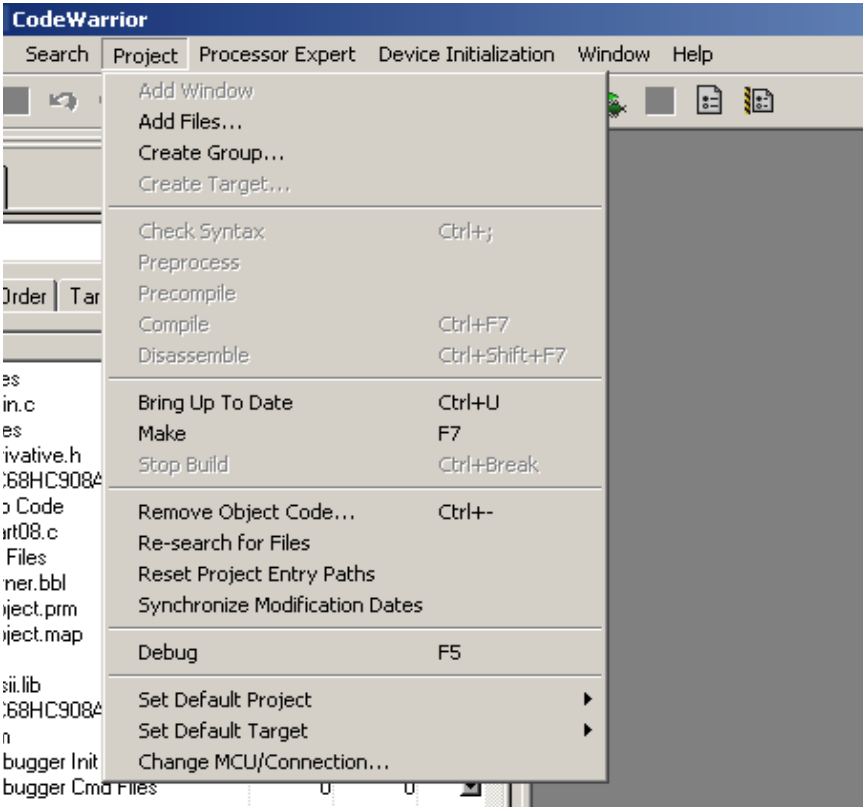
To start the debugger from the **Project** window, click the **Debug** icon (Figure 2.1 on page 8), at the top of the Project window.

Figure 2.1 Project Window Make and Debug Icons

Starting Debug from the Main Window Menu Bar

You can also start the debugger from the main menu bar of the CodeWarrior IDE. To start the debugger from the main menu bar, select Debug from the Project menu:
(**Project** > **Debug**.)

Figure 2.2 Main Window Project Menu



Debugger Command Line Start

You can start the debugger from a DOS command line. The command syntax is as follows:

```
HIWAVE.EXE [<AbsFileName> {-<options>}]
```

where **`AbsFileName`** is the name of the application to load in the debugger. Preceed each option with a dash.

Command Line Options

DOS command line options are:

-T=<time>: Test mode

The debugger terminates after the specified time (in seconds). The default value is 300 seconds. For example:

```
c:\Freescale\prog\hiwave.exe -T=10
```

The above example instructs the debugger to terminate after 10 seconds.

-Target=<targetname>

This option sets the specified connection. For example:

```
C:\Freescale\prog\hiwave.exe
c:\Freescale\demo\hc12\sim\fibo.abs -w -Target=sim
```

The command in the above example starts the debugger and loads fibo.abs file.

-W: Wait mode

Debugger will wait even when a <exeName> is specified.

-Instance=%currentTargetName

This option defines a build instance name. When a build instance is defined, the same one will be used. For example:

```
c:\Freescale\prog\hiwave.exe -Instance=%currentTargetName
```

If you attempt to start the debugger again, the existing instance of the debugger is brought to the foreground.

-Prod= <fileName>

This option specifies the project directory and/or project file to be used at start-up. For example:

```
c:\Freescale\prog\hiwave.exe -Prod=c:\demoproject\test.pjt
```

-Nodefaults

Debugger will not load the default layout (see section 4 of the Project file Activation). For example:

```
c:\Freescale\prog\hiwave.exe -nodefaults
```

-Cmd = <Command>

This option specifies a command to be executed at start-up: -cmd = "" {characters}. For example:

```
c:\Freescale\prog\hiwave.exe -cmd="open recorder"
```

-C <cmdFile>

This option specifies a command file to be executed at start-up. For example:

```
c:\Freescale\prog\hiwave.exe -c c:\temp\mycommandfile.txt
```

-ENVpath: "-Env" <Environment Variable> "=" <Variable Setting>

This option sets an environment variable. This environment variable may be used to overwrite system environment variables. For example:

```
c:\Freescale\prog\hiwave.exe -EnvOBJPATH=c:\sources\obj
```

NOTE Options are not case sensitive.

Order of Commands

Commands specified by options are executed in the following order:

1. Load (activate) the project file (see below). If the project file is not specified, “project.ini”is used by default.
2. Load <exeFile> if available and start* running unless option |(W) was specified
3. Execute command file <cmdFile> if specified
4. Execute command if specified
5. *Start running unless option |(W) was specified

NOTE * In version 6.0 of the debugger, the loaded program is started after all command and command files are executed.

NOTE The function **Open** in the File menu will interpret any file without an `.ini` extension as a command file and not a project file.

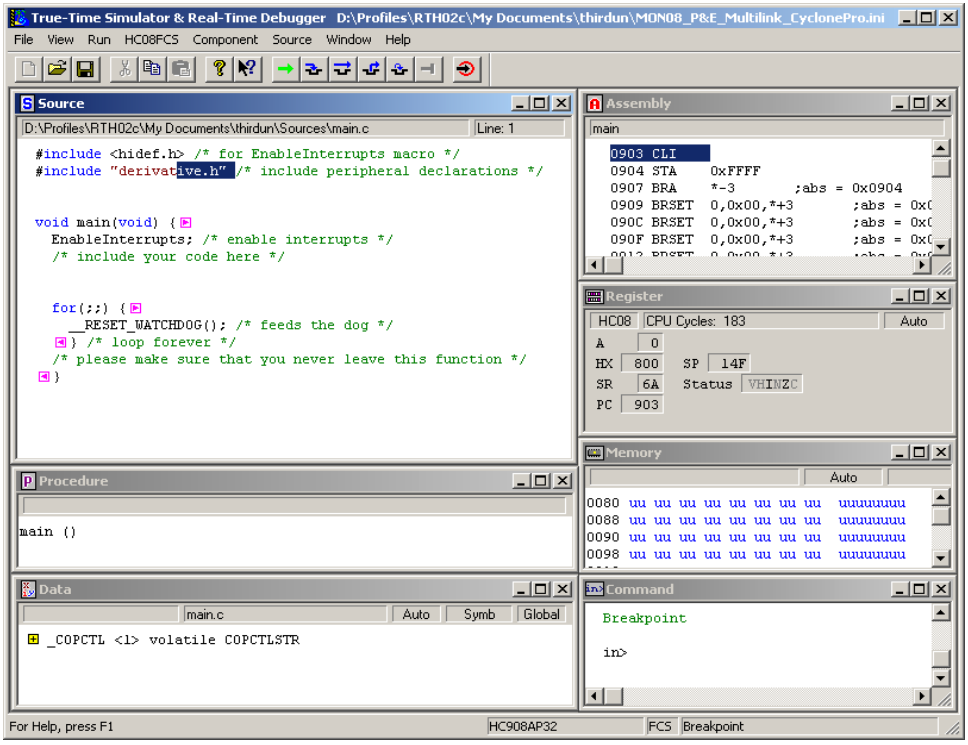
Example

```
C:\Freescale\PROG \DEMO\TEST.ABS -w -d
```

Debugger Main Window

Once you start the Debugger, the True Time Simulator & Real Time Debugger window opens in the right side of the IDE Main Window.

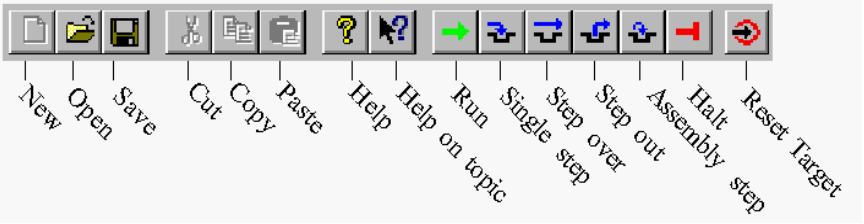
Figure 2.3 Debugger Main Window



Debugger Main Window Toolbar

The Debugger Main Window toolbar is the default toolbar. Most of the Main Window menu commands have a related shortcut icon on this toolbar. Figure 2.4 on page 13 identifies each default icon.

Figure 2.4 Debugger Main Window Toolbar



A tool tip is available when you point the mouse at an icon.

Debugger Main Window Status Bar

The status bar at the bottom of the Debugger Main Window, shown in Figure 2.5 on page 13 contains a context sensitive help line for connection specific information, e.g., number of CPU cycles for the **Simulator** connection and execution status.

Figure 2.5 The Debugger Status Bar

Main Window Menu Bar

The Debugger Main Window Menu Bar, shown in Figure 2.6 on page 13 is associated with the main function of the debugger application, connection, and selected windows.

Figure 2.6 Debugger Window Menu Bar

True-Time Simulator & Real-Time Debugger D:\Profiles\RTH02c\My Documents\th
File View Run HC08FCS Component Source Window Help

NOTE You can select menu commands by pressing the ALT key to select the menu bar. then pressing the key corresponding to the underlined letter in the menu command.

Table 2.1 on page 14 describes menu entries available in the menu bar.

Table 2.1 Description of the Main Menu Toolbar Entries

Menu entry	Description
File	Contains entries to manage debugger configuration files.
View	Contains entries to configure the toolbar.
Run	Contains entries to monitor a simulation or debug session.
Connection	Contains entries to select the debugger connection. Once a connection has been selected, the name of this heading changes.
Component	Contains entries to select and configure extra component window
Data	Contains entries to select Data component functions.
Window	Contains entries to set the component windows.
Help	A standard Windows Help menu.

File Menu

The **File** menu shown in Figure 2.7 on page 15 is dedicated to the debugger project.

Figure 2.7 File Menu

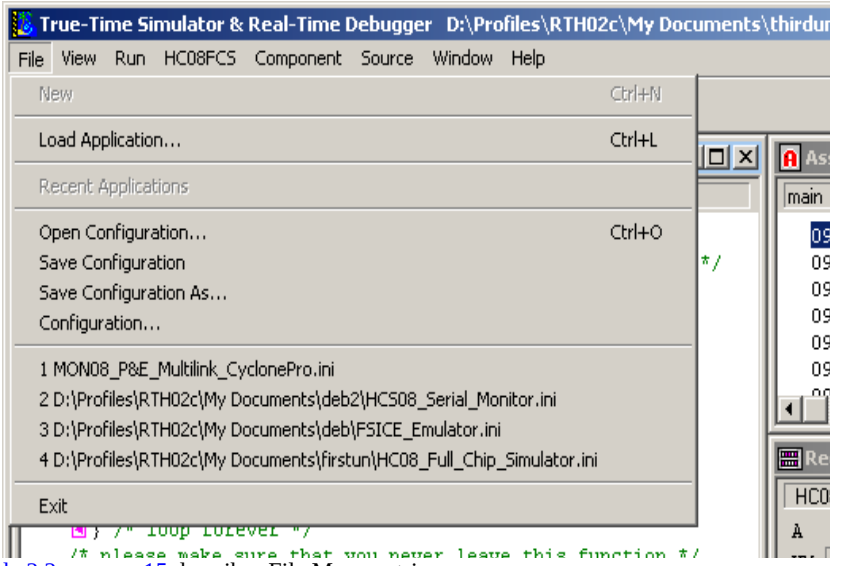


Table 2.2 on page 15 describes File Menu entries.

Table 2.2 File Menu Entry Description

Menu Entry	Description
New	Creates a new project.
Load Application	Loads an executable file (or debugger connection if nothing is selected).
...\restart.abs ...\await.abs ...	Recent applications list
Open Configuration	Opens the debugger project window. You can load a project file .**PJT** or .**INI**. Additionally you can load an existing .**HWC** file corresponding to a debugger configuration file. You can load a project .**INI** file containing component names, associated window positions and parameters, window parameters (fonts, background colors, etc.), connection name e.g., **Simulator** and the .**ABS** application file to load.
Save Configuration	Saves the project file

Table 2.2 File Menu Entry Description

Menu Entry	Description
Save Project As	Opens the debugger project window to save the project file under a different path and name, and format (PJT; INI...).
Configuration	Opens the Preferences window to set environment variables for current project.
1.Project.ini 2.Test.ini 3...	Recent project file list
Exit	Quits the Debugger.

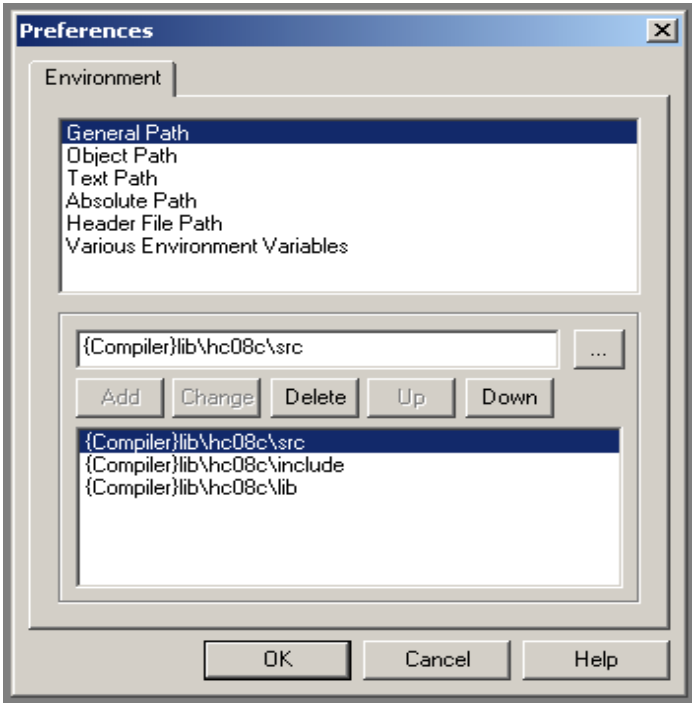
You can shortcut some of these functions by clicking toolbar icons (refer to the Debugger Main Window Toolbar on page 12 section).

Preferences Window

Open the Preferences window by selecting **Configuration** from the **Files** menu. With this window (Figure 2.9 on page 18) it is possible to set up environment variables for the current project. New variables are saved in the current project file when you clickthe **OK** button.

NOTE The corresponding menu entry (`File>Configuration`) is only enabled if a project file is loaded.

Figure 2.8 Preferences Window - Environment Tab



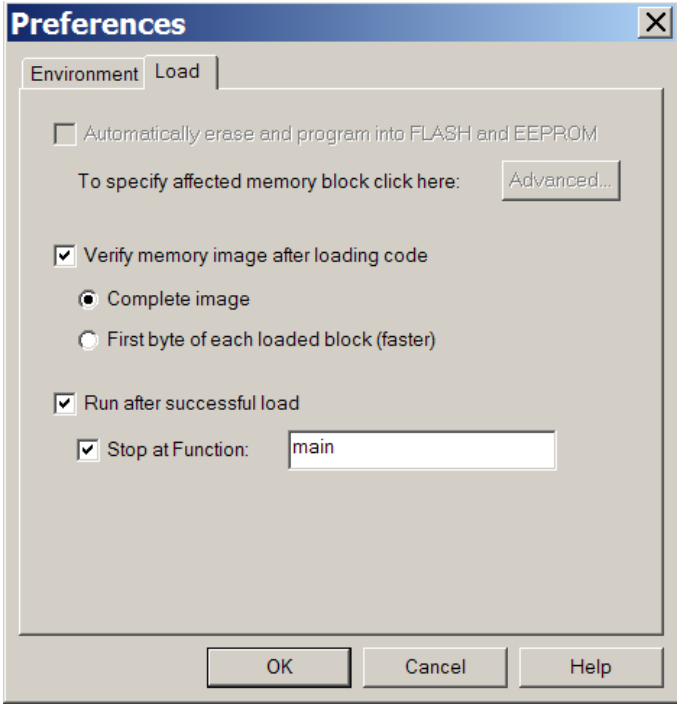
The **Preferences** Window - Environment tab contains the following controls:

- A list box containing all available environment variables. You can select a variable with the mouse or Up/Down buttons.
- Command Line Arguments are displayed in the text box. You can add, delete, or modify options, and specify a directory with the browse button (...).
- A second list box contains the arguments for all of the environment variables defined in the corresponding Environment section. Select a variable with the mouse or Up/Down buttons.

Command Buttons:

- **OK**: Changes are confirmed and saved in current project file.
- **Cancel**: Closes dialog box without saving changes.
- **Help**: Opens the help file.

Figure 2.9 Preferences Window - Load Tab



The **Preferences** Window - Load tab contains the following controls:

- An Automatically erase and program into FLASH and EEPROM checkbox.
- A Verify memory image after loading code checkbox, with two radio buttons that let you define the memory image.
- Run after successful load checkbox.
- A Stop at Function checkbox with a textbox that lets you define the function. Command Buttons:
- **OK**: Changes are confirmed and saved in current project file.
- **Cancel**: Closes dialog box without saving changes.
- **Help**: Opens the help file.

View Menu

In the Main Window View menu (Figure 2.10 on page 19) you can choose to show or hide the toolbar, status bar, window component titles and headlines (see the Component Windows Object Info Bar on page 35). You can select smaller window borders and customize the toolbar. Table 2.3 on page 19 describes the View Menu entries.

Figure 2.10 View Menu

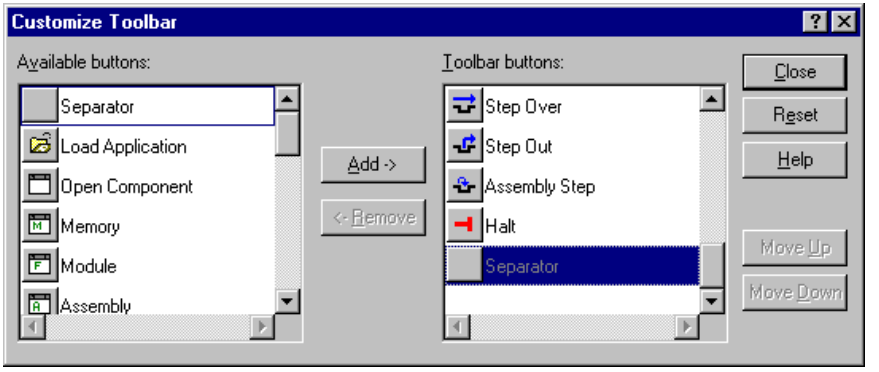
Table 2.3 View Menu Description

Menu Entry	Description
Toolbar	Check / uncheck Toolbar if you want to display or hide it.
Status Bar	Check / uncheck Status Bar if you want to display or hide it.
Hide Tile	Check / uncheck Hide Title if you want to hide or display the window title.
Hide Headline	Check / uncheck Hide Headline if you want to hide or display the headline.
Small Borders.	Check / uncheck Small Border if you want to display or hide small window borders.
Customize	Opens the debugger Customize Toolbar window.

Customizing the Toolbar

When you select **Customize** from the **View** menu, the Customize Toolbar dialog box appears. You can customize the toolbar of the Debugger, adding and removing component shortcuts and action shortcuts in this dialog box. You can also insert separators to separate icons. Almost all functions in **View**, **Run** and **Window** menus are available as shortcut buttons, as shown in Figure 2.11 on page 20.

Figure 2.11 Customize Toolbar Dialog Box



- Select the desired shortcut button in the **Available buttons** list box and click **Add** to install it in the toolbar.
- Select a button in the **Toolbar buttons** list box and click **Remove** to remove it from the toolbar.

Demo Version Limitations

The default toolbar cannot be configured.

Examples of View Menu Options

Figure 2.12 on page 20 shows a typical component window display.

Figure 2.12 Typical Component Window Display

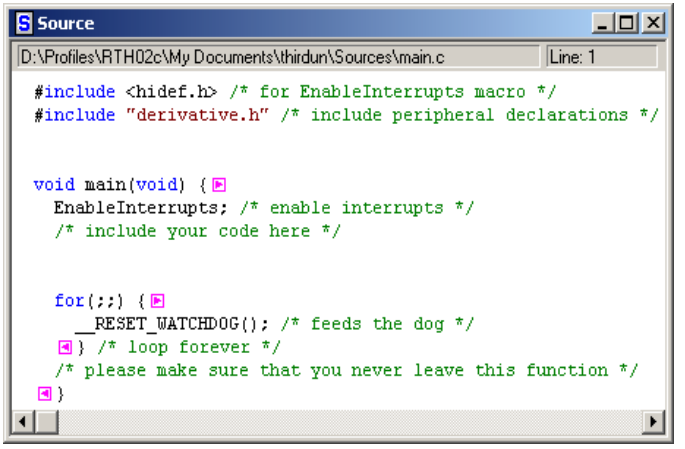

Source
D:\Profiles\RTH02c\My Documents\thirdun\Sources\main.c
Line: 1
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
void main(void) {
EnableInterrupts; /* enable interrupts */
/* include your code here */
for(;;) {
__RESET_WATCHDOG(); /* feeds the dog */
} /* loop forever */
/* please make sure that you never leave this function */
}

Figure 2.13 on page 21 shows a component window without a title and headline.

Figure 2.13 Component Window without Title and Headline

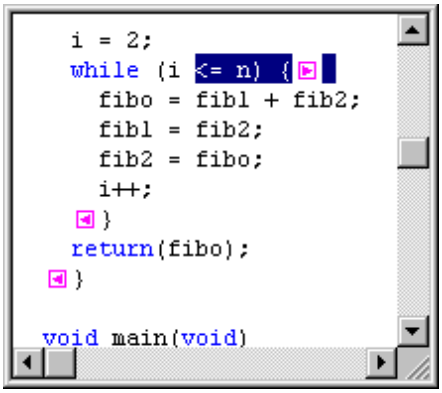


Figure 2.14 on page 21 shows a component window without a title and headline, and with a small border.

Figure 2.14 Component Window without Title and Headline, and with Small Border

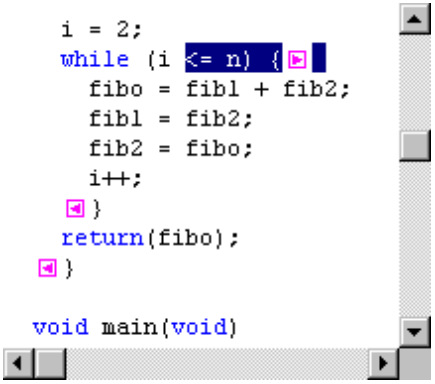
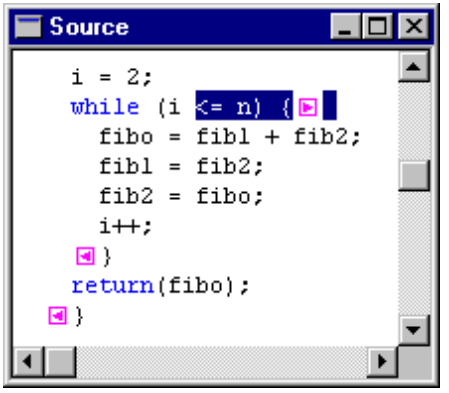


Figure 2.15 on page 22 shows a component window without headline and small border

Figure 2.15 Component Window without Headline and Small Border



Run Menu

The Main Window Run menu, shown in Figure 2.16 on page 22 is associated with the debug session. You can monitor a simulation or debug session from this menu. Run menu entries are described in Table 2.4 on page 23.

Figure 2.16 Run Menu

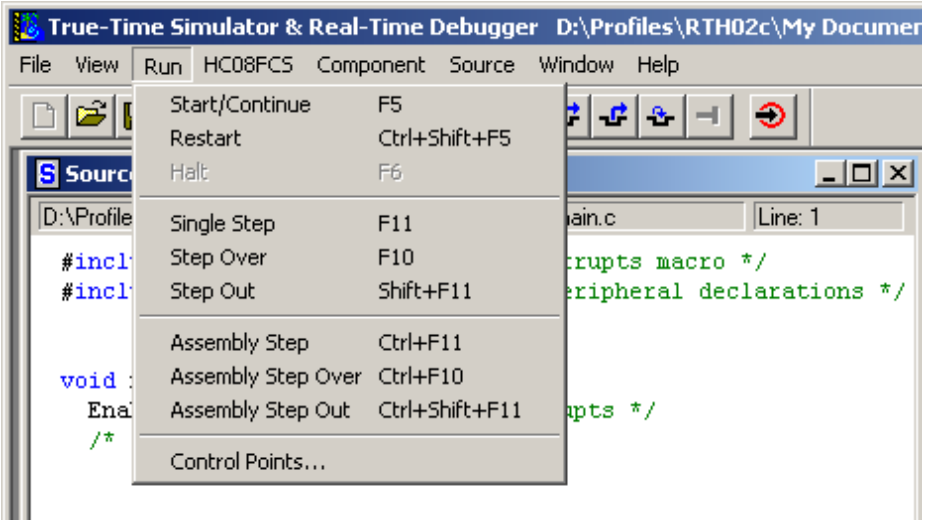


Table 2.4 Run Menu Description

Menu entry	Description
Start/Continue	Starts or continues execution of the loaded application from the current program counter (PC) until a breakpoint or watchpoint is reached, runtime error is detected, or user stops the application by selecting Run -> Halt. Shortcut: **F5 key**
Restart	Starts execution of the loaded application from its entry point. Shortcut: **CTRL + Shift + F5 keys**
Halt	Interrupts and halts a running application. You can examine the state of each variable in the application, set breakpoints, watchpoints, and inspect source code. Shortcut: **F6 key**
Single Step	If the application is halted, this command performs a single step at the source level. Execution continues until the next source reference is reached. If the current statement is a procedure call, the debugger “steps into” that procedure. The **Single Step** command does not treat a function call as one statement, therefore it steps into the function. Shortcut: **F11 key**
Step Over	Similar to the **Single Step** command, but does not step into called functions. A function call is treated as one statement. Shortcut: **F10 key**
Step Out	If the application is halted inside of a function, this command continues execution and then stops at the instruction following the current function invocation. If no function calls are present, then the **Step Out** command is not performed. Shortcut: **Shift + F11 keys**
Assembly Step	If the application is halted, this command performs a single step at the assembly level. Execution continues for one CPU instruction from the point it was halted. This command is similar to the Single Step command, but executes one machine instruction rather than a high level language statement. Shortcut: **CTRL + F11 keys**

Table 2.4 Run Menu Description (*continued*)

Menu entry	Description
Assembly Step Over	Similar to the **Step Over** command, but steps over subroutine call instructions. Shortcut: **CRTL + F10 keys**
Assembly Step Out	If the application is halted inside a function, this command continues execution and stops on the CPU instruction following the current function invocation. This command is similar to the **Step Out** command, but stops before the assignment of the result from the function call. Shortcut: **CTRL + Shift + F11 keys**
Control Points	Opens the Controlpoints Configuration Window which contains tabs that allow you to control Breakpoints, Watchpoints and Markpoints (refer to Control Points on page 145 chapter).

You can provide shortcuts for some of these functions using the toolbar. Refer to the Debugger Main Window Toolbar on page 12 and Customizing the Toolbar on page 19 sections for details.

You can also set breakpoints and watchpoints from within the Source and Assembly component windows.

NOTE For more information about breakpoints and watchpoints, refer to the Control Points on page 145 chapter.

Connection Menu

This menu entry (Figure 2.17 on page 25) appears between the **Run** and **Component** menus when no connection is specified in the `PROJECT.INI` file and no connection has been set. The **Connection** name is replaced by an actual connection name when the connection is set. If a connection has been set, the number of menu entries is expanded, depending on the connection. To set the connection, select **Component>Set Connection...** Refer to the Component Menu on page 29 section for details.

Figure 2.17 Connection Menu

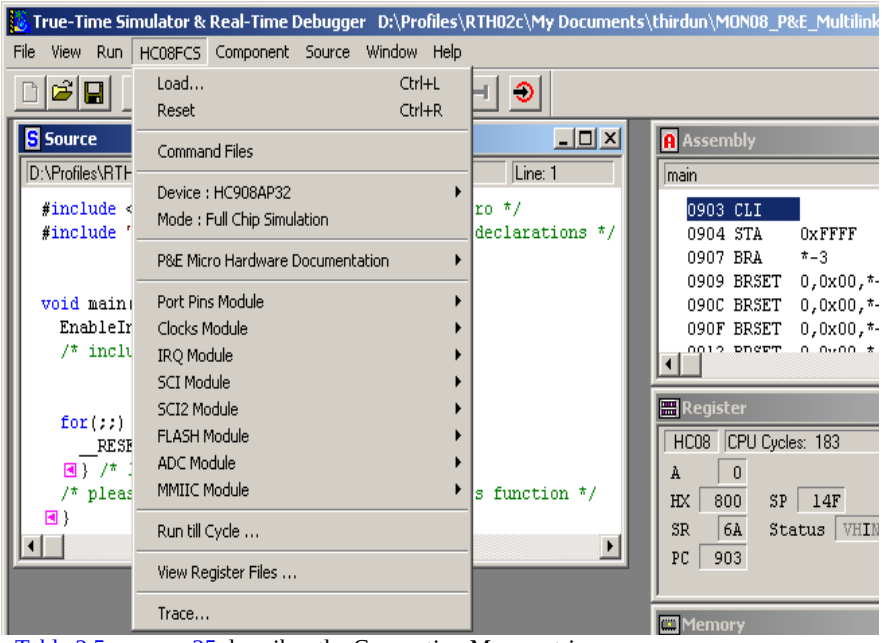


Table 2.5 on page 25 describes the Connection Menu entries.

Table 2.5 Connection Menu Common Option Description

Menu Entry	Description
Load	Loads a connection.
Reset	Resets the current connection.

Loading a Connection

Use the Connection menu to load a debugger connection.

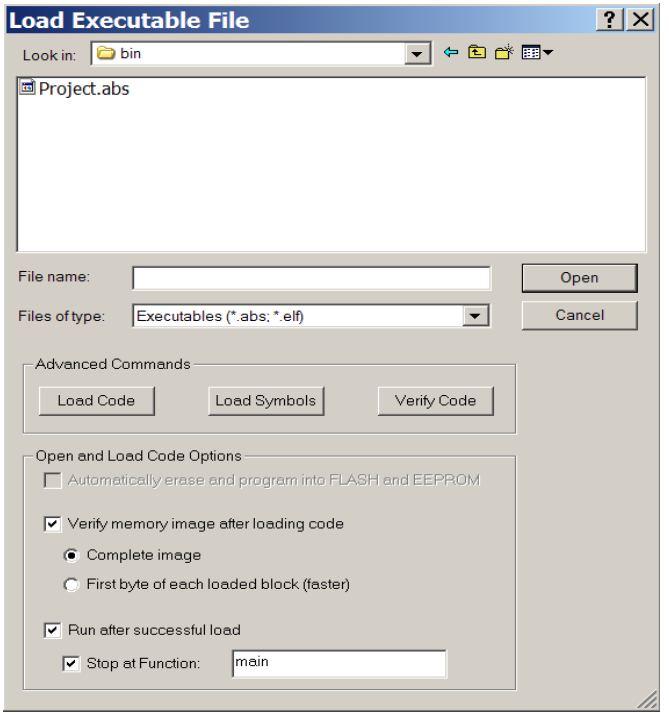
1. Choose **Connection>Load...**

   The Load Executable File window shown in Figure 2.18 on page 26 is displayed:

Load Executable File Window

From the Connection menu, choose **Load...** to open the Load Executable File window, shown in Figure 2.18 on page 26, then set the load options and choose a Simulation Execution Framework (an `.ABS` application file).

Figure 2.18 Load Executable File Window



Open Button

When this button is pressed, the application code and symbols are loaded.

Load Options Buttons

These three buttons allow you to select which part of the executable file will be loaded:

- **Load Code Button**: Loads the application code only. Only the application is loaded into the target system. This button can be used if no debugging is needed.
- **Load Symbols Button:** Loads symbols only. Only debugging information is loaded. This button can be used if the code is already loaded into the target system or programmed into a non-volatile memory device (ROM/FLASH).
- **Verify Code Button:** Loader loads no data into memory. However, it reads back current data matching the same areas from the target memory and compares all data with the data from the selected file.

Open and Load Code Options Area

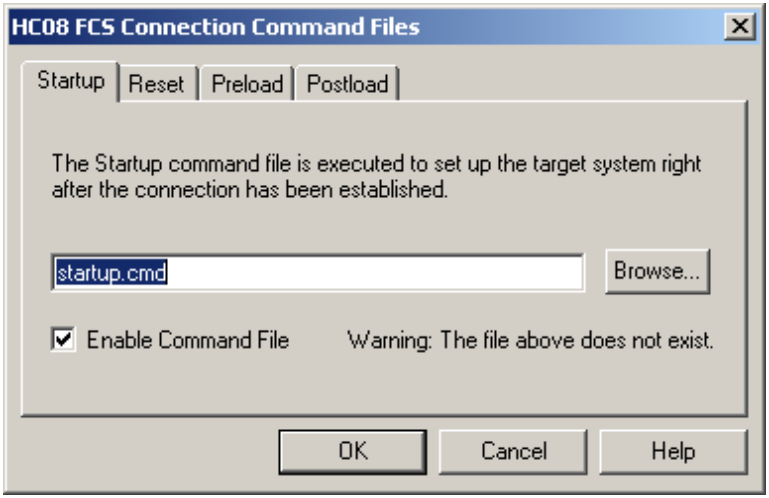
The checkboxes and buttons of this area of the Load Executable File window offer the dollowing options:

- An Automatically erase and program into FLASH and EEPROM checkbox.
- A Verify memory image after loading code checkbox, with two radio buttons that let you define the memory image.
- Run after successful load checkbox.
- A Stop at Function checkbox with a textbox that lets you define the function. Command Buttons:
- **OK**: Changes are confirmed and saved in current project file.
- **Cancel**: Closes dialog box without saving changes.
- **Help**: Opens the help file.

Connection Command File Window

From the Connection menu, choose **Command File** to open the Connection Command File window. Each tab of this window, shown in Figure 2.19 on page 27 corresponds to an event on which a command file can be automatically run from the . See the Startup Command File on page 28, Reset Command File on page 28, Preload Command File on page 29, and Postload Command File on page 29, sections that follow.

Figure 2.19 Connection Command File Window



The command file in the edit box is executed when the corresponding event occurs.Click the **Browse** button to set the path and name of the command file.

The **Enable Command File** check box allows you to enable/disable a command file on an event. By default, all command files are enabled:

- The default **Startup** command file is `STARTUP.CMD`,
- The default **Reset** command file is `RESET.CMD`,
- The default **Preload** command file is `PRELOAD.CMD`,
- The default **Postload** command file is `POSTLOAD.CMD`.

NOTE **Startup** settings performed in this dialog are stored for subsequent debugging sessions in the **[Simulator]** section of the **PROJECT** file using the variable **CMDFILE0.**

NOTE When a CPU is set, the settings performed in this dialog are stored for subsequent debugging sessions in the **[Simulator XXX]** (where XXX is the processor) section of the **PROJECT** file using variables **CMDFILE0, CMDFILE1,... CMDFILEn**.

Startup Command File

The **Startup** command file is executed by the after the connection has been loaded.

The **Startup** command file full name and status (enable/disable) can be specified either with the **CMDFILE STARTUP** Command Line command or using the **Startup** property tab of the Connection Command File Window on page 27.

By default the `STARTUP.CMD` file located in the current project directory is enabled as the current **Startup** command file.

Reset Command File

The **Reset** command file is executed by the after the reset button, menu entry or Command Line command has been selected.

The **Reset** command file full name and status (enable/disable) can be specified either with the **CMDFILE RESET** Command Line command or using the **Reset** property tab of the Connection Command File Window on page 27.

By default the `RESET.CMD` file located in the current project directory is enabled as the current **Reset** command file.

Preload Command File

The **Preload** command file is executed by the before an application is loaded to the target system through the connection.

The **Preload** command file full name and status (enable/disable) can be specified either with the **CMDFILE PRELOAD** Command Line command or using the **Preload** property tab of the Connection Command File Window on page 27.

By default the `PRELOAD.CMD` file located in the current project directory is enabled as the current **Preload** command file.

Postload Command File

The **Postload** command file is executed by the after an application has been loaded to the target system through the connection.

The **Postload** command file full name and status (enable/disable) can be specified either with the **CMDFILE POSTLOAD** Command Line command or using the **Postload** property tab of the Connection Command File Window on page 27.

By default the `POSTLOAD.CMD` file located in the current project directory is enabled as the current **Postload** command file.

Component Menu

The Component menu is shown in Figure 2.19, Component Menu on page 29.

Figure 2.20 Component Menu

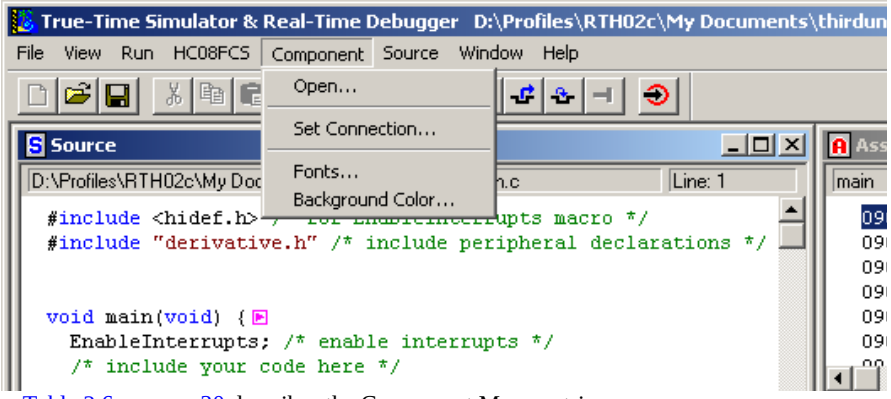


Table 2.6 on page 30 describes the Component Menu entries.

Table 2.6 Component Menu Description

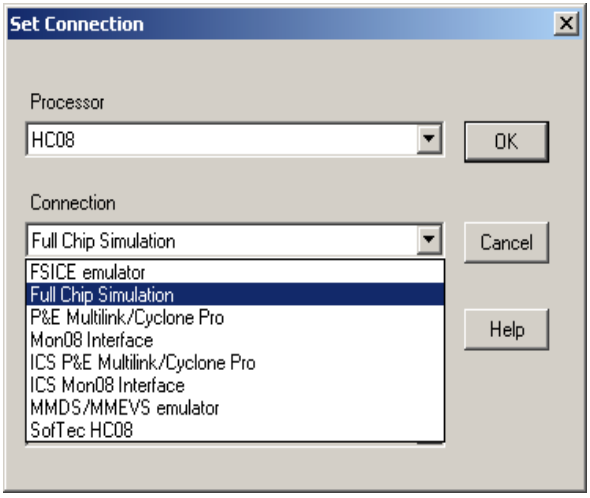
Menu entry	Description
Open	Loads an extra component window that has not been loaded by the Debugger at startup. The popup dialog presents a set of different components that are introduced in the Typical Component Window Display on page 20 section.
Set Connection	Sets the Debugger connection.
Fonts	Opens a standard Font Selection dialog, where you can set the font used by Debugger components.
Background Color	Opens a standard Color Selection dialog, where you can set the background color used by the Debugger component windows.

NOTE For a readable display, we recommend using a proportional font (e.g., Courier, Terminal, etc.).

Select **Component>Open...** to load an extra component window that has not been loaded by the Debugger at startup. The popup dialog presents a set of different components that are introduced in Debugger Components on page 45.

Select **Component>Set Connection...** and the **Set Connection** dialog box shown in Figure 2.21 on page 31 is opened.

Figure 2.21 Set Connection Dialog Box



2. Use the **Processor** list popup to select the desired processor.
3. Use the **connection** list popup to select the desired connection.

   A text panel displays information about the selected connection.

NOTE When a connection cannot be loaded, the combo box displays the path where you should install missing dll.

4. Click **OK** to load connection in debugger.

NOTE For more information about which connection to load and how to set/reset a connection, refer to the Debugger connection books in Sections II and III of this manual.

Window Menu

In this menu, shown in Figure 2.22 on page 32, you can set the component windows general arrangement. The Submenu **Window>Options** is shown in Figure 2.23 on page 32 and the Submenu **Window>Layout** in Figure 2.24 on page 32.

Figure 2.22 Window Menu

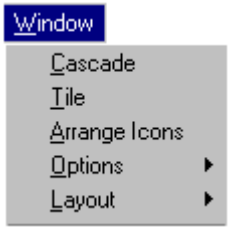


Figure 2.23 Window Menu Options SubMenu

Figure 2.24 Window Menu Layout SubMenu

Table 2.7 on page 32 specifies the Window Menu entries.

Table 2.7 Window Menu Description

Menu entry	Description
Cascade	Option to arrange all open windows in cascade (so they overlap).
Tile	Option to display all open windows in tile format (non overlapping).
Arrange Icons	Arranges icons at the bottom of windows.
Options - Autosize	Component windows always fit into the debugger window whenever you modify the debugger window size.
Options - Component Menu	When a component window is selected, the associated menu is displayed in the main menu. For example if you select the Source window, the Source menu is displayed in the main menu.
Layout - Load/Store	Option to **Load / Store** your arrangements from a `.HWL` file.

NOTE Autosize and Component Menu are checked by default.

Help Menu

This is the Debugger Main window Help menu (Figure 2.25 on page 33). Table 2.8 on page 33 shows menu entries.

Figure 2.25 Help Menu

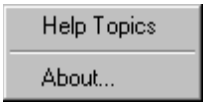


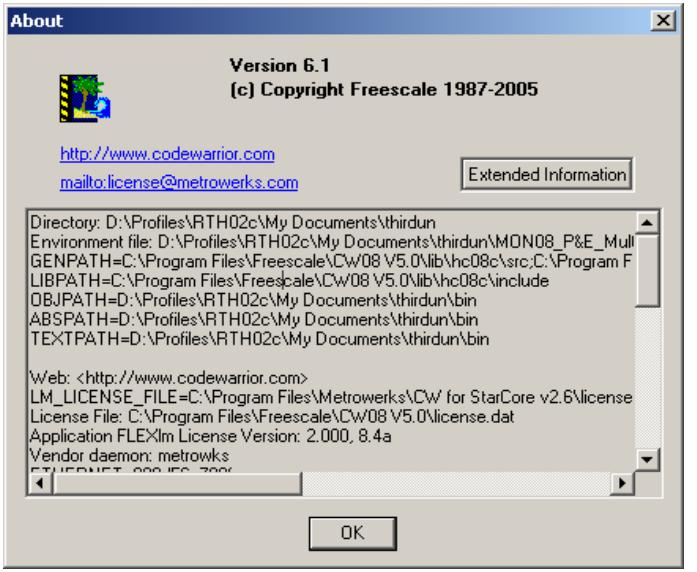
Table 2.8 Help Menu Description

Menu entry	Description
Help Topics	Choose **Help Topics** in the menu for online help or if you need specific information about a topic.
About ...	Information about the debugger version and copyright, and license information is displayed.

About Box

Select **Help>about** to display the About box, shown in Figure 2.26 on page 34. The about box lists directories for the current project, system information, program information, version number and copyright. It contains information to send for Registration: you can copy this information and send to `license@Freescale.com`.

Figure 2.26 About Box



For more information on all components, click on the **Extended Information** button.

Two hypertext links allow you to send an E-mail for a license request or information, and open the Freescale internet home page.

Click on **OK** to close this dialog box.

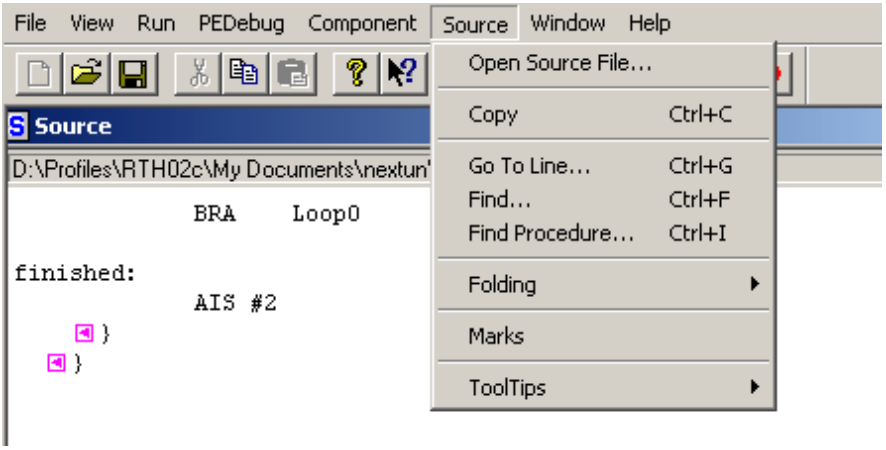
Component Associated Menus

Vaious Debugger Component windows are shown in Figure 2.3 on page 12. Each component window loaded by default or that you have loaded has two menus. One menu is in the main menu and the other one is a popup menu (also called “Associated Popup Menu”) that you can open by right-clicking in a window component. Note that before right-clicking, the component window has to be active.

Component Main Menu

This menu, shown in Figure 2.27 on page 35 is always between the Component entry and the Window entry of the Debugger main window toolbar. It contains general entries of the current active component. You can hide this menu by unchecking **Window>Options>Component Menu.**

Figure 2.27 Example of Component Main Menu



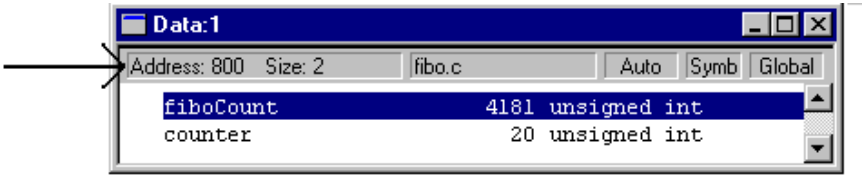
Component Files

Each component is a windows file with a **.wnd** extension

Component Windows Object Info Bar

The object info bar of the debugger window, as shown in Figure 2.28 on page 35, provides information about the selected object.

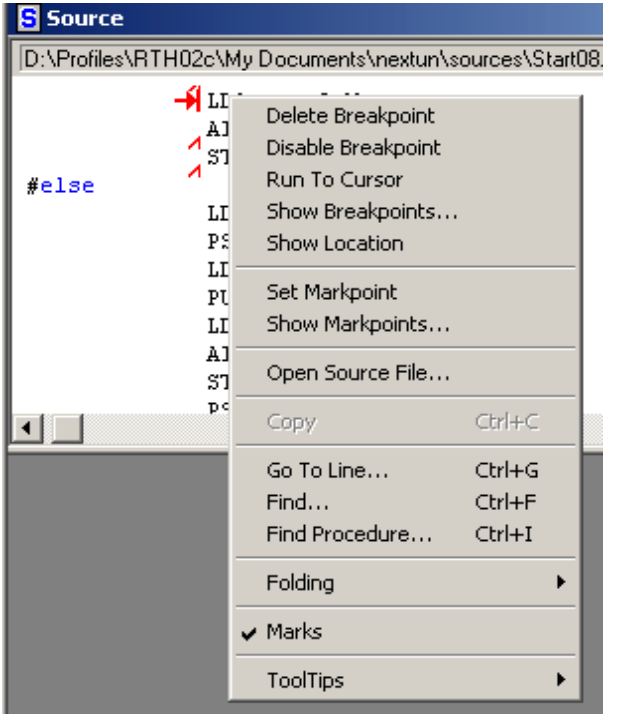
Figure 2.28 Object Info Bar of Debugger Component Windowss



Component Popup Menu

The popup menu is a dynamic context sensitive menu. It contains entries for additional facilities available in the current component. Depending on the position of the mouse in the window and what is being pointed to, popup menu entries will differ.

Figure 2.29 Example of Component Popup Menu



For example, if you click the mouse on a breakpoint, menu options allow you to delete, enable, or disable the breakpoint.

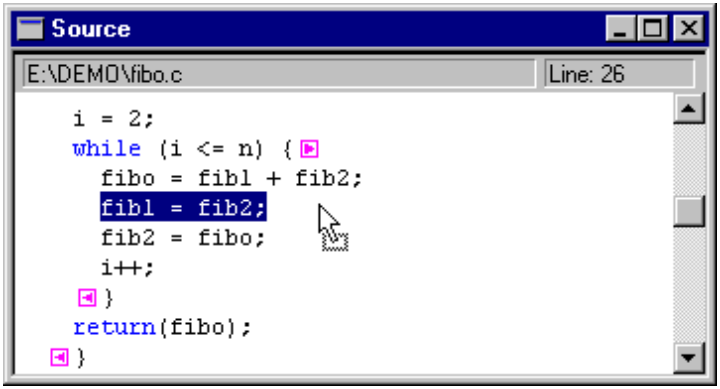
Highlights of the User Interface

This section describes some of the the main features of the Debugger user interface.

Activating Services with Drag and Drop

You can activate services by dragging objects from one component window to another. This is known as drag and drop, an example is shown in Figure 2.30 on page 37.

Figure 2.30 Drag and Drop Example

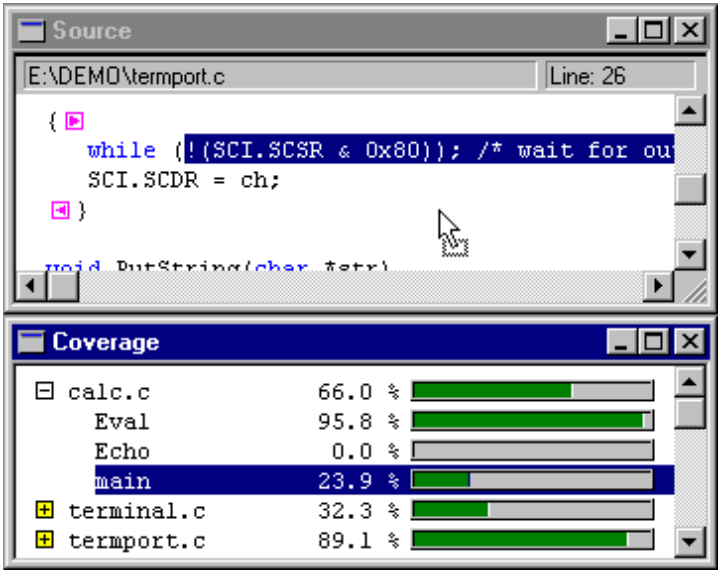


When the destination of a dragged item is not possible, the following cursor symbol is displayed:

Example:

You can activate the display of coverage information on assembler and C statements by dragging the chosen procedure name from the Coverage component to the Source and Assembly components (Figure 2.31 on page 38).

Figure 2.31 Dragging Procedure Name from Coverage to Source Component Window.



You can display the memory layout corresponding to the address held in a register by dragging the address from the Register Component to the Memory Component.

To Drag and Drop an Object

To drag an object from one component window to another:

1. Select the component containing the object you want to drag.
2. Make sure the destination component window where you want to drag the object is visible.
3. Select the object you want.
4. Pressing and holding the left mouse button, drag the object into the destination component window and then release the mouse button.

Drag and Drop Combinations

Dragging and dropping objects is possible between different component windows and are introduced in each component description section.

See below, the possible combinations of drag and drop between components and associated actions. When additional components are available, new combinations might be possible and described in the component's information manual.

Dragging from Assembly Component Window

Table 2.9 on page 39 summarizes dragging from the Assembly Component.

Table 2.9 Dragging from the Assembly Component Window

Destination Component Window	Action
Command Line	The Command Line component appends the address of the "pointed to" instruction to the current command.
Memory	Dumps memory starting at the selected instruction PC. The PC location is selected in the memory component.
Register	Loads the destination register with the PC of the selected instruction.
Source	Source component scrolls up to the source statements and highlights it.

Dragging from Data Component Window

Table 2.10 on page 40 summarizes dragging from the Data Component.

Table 2.10 Dragging from the Data Component Window

Destination Component Window	Action
Command Line	Dragging the name appends the address range of the variable to the current command in the Command Line Window. Dragging the value appends the variable value to the current command in the Command Line Window.
Memory	Dumps memory starting at the address where the selected variable is located. The memory area where the variable is located is selected in the memory component.
Register	Dragging the name loads the destination register with the address of the selected variable. Dragging the value loads the destination register with the value of the variable.
Source	Dragging the name of a global variable in the source Windows display the module where the variable is defined and the source text is searched for the first occurrence of the variable and highlighted.

NOTE It is not possible to drag an expression defined with the Expression Editor. The "forbidden" cursor is displayed.

Dragging from Source Component Window

Table 2.11 on page 40 summarizes dragging from the Source Component.

Table 2.11 Dragging from the Source Component Window

Destination Component Window	Action
Assembly	Displays disassembled instructions starting at the first high level language instruction selected. The assembler instructions corresponding to the selected high level language instructions are highlighted in the Assembly component
Register	Loads the destination register with the PC of the first instruction selected.

Table 2.11 Dragging from the Source Component Window

Destination Component Window	Action
Memory	Displays the memory area corresponding with the high level language source code selected. The memory area corresponding to the selected instructions are greyed in the memory component.
Data	A selection in the Source window is considered an expression in the Data window, as if it was entered through the Expression Editor of the Data component. (please see Data Component on page 64 and Expression Editor on page 65)

Dragging from the Memory Component Window

Table 2.12 on page 41 summarizes dragging from the Memory Component.

Table 2.12 Dragging from the Memory Component Window

Destination Component Window	Action
Assembly	Displays disassembled instructions starting at the first address selected. Instructions corresponding to the selected memory area are highlighted in the Assembly component.
Command Line	Appends the selected memory range to the Command Line window
Register	Loads the destination register with the start address of the selected memory block.
Source	Displays high level language source code starting at the first address selected. Instructions corresponding to the selected memory area are greyed in the source component.

Dragging from Procedure Component Window

Table 2.13 on page 42 summarizes dragging from the Procedure Component.

Table 2.13 Dragging from the Procedure Component Window

Destination Component Window	Action
Data > Local	Displays local variables from the selected procedure in the data component
Source	Displays source code of the selected procedure. Current instruction inside the procedure is highlighted in the Source component.
Assembly	The current assembly statement inside the procedure is highlighted in the Assembly component.

Dragging from Register Component Window

Table 2.14 on page 42 summarizes dragging from the Register Component Window.

Table 2.14 Dragging from the Register Component Window

Destination Component Window	Action
Assembly	Assembly component receives an address range, scrolls to the corresponding instruction and highlights it.
Memory	Dumps memory starting at the address stored in the selected register. The corresponding address is selected in the memory component.

Dragging from Module Component Window

Table 2.15 on page 43 summarizes dragging from the Register Component.

Table 2.15 Dragging from the Module Component Window

Destination Component Window	Action
Data > Global	Displays global variables from the selected module in the data component
Memory	Dumps memory starting at the address of the first global variable in the module. The memory area where this variable is located is selected in the memory component.
Source	Displays source code from selected module.

Selection Dialog Box

This dialog box is used in the Debugger for opening general components or source files. You can select the desired item with the arrow keys or mouse and then the **OK** button to accept or **CANCEL** to ignore your choice. The **HELP** button opens this section in the Help File.

This dialog box is used for the following selections:

- Set Connection
- Open IO component
- Open Source File
- Open Module
- Individual component window

3

Debugger Components

This chapter explains how the different components of the Debugger work. This chapter contains the following sections:

- Component Introduction on page 45
- Loading Component Windows on page 46
- General Debugger Components on page 48
- Visualization Utilities on page 121

Component Introduction

The Debugger kernel includes various components.

CPU Components

CPU components handle processor specific properties such as register naming, instruction decoding (disassembling), stack tracing, etc. A specific implementation of the CPU module has to be provided for each processor type that is supported in the debugger. The CPU related component is not introduced in this section. However, this system component is reflected in the Register component, Memory component, and all other Connection dependent components. The appropriate CPU component is automatically loaded when loading a framework (.**ABS** file). Therefore it is possible to mix frameworks for different MCUs. The Debugger automatically detects the MCU type and loads the appropriate CPU component, if available on your environment.

Window Components

The Debugger main window components are small applications loaded into the debugger framework at run-time. Window components can access all global facilities of the debugger engine, such as the connection (to communicate with different connections), and the symbol table. The Debugger window components are implemented as dynamic link libraries (DLLs) with extension.**WND**. These components are introduced in this section.

Connection Components

Different debugger connections are available. For example, you can set a CPU awareness to simulate your `.ABS` application files, and also set a background debugger.

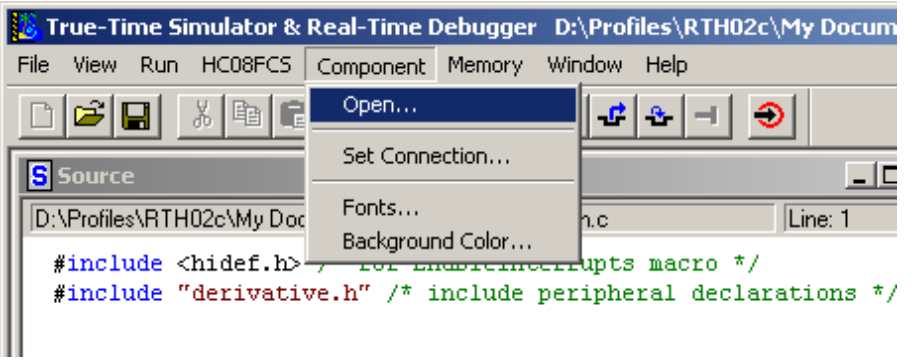
Different connections are available to connect the target system (hardware) to the debugger. For example, the connection may be connected using a Full Chip Simulator, an Emulator, a ROM monitor, a BDM pod cable, or any other supported device.

NOTE Connection components are introduced in their respective manuals.

Loading Component Windows

In the Debugger Main Window Menu Bar, shown in Figure 3.1 on page 46, you can use the Component menu to load all framework components. Each Debugger component you select will appear as a window in the Debugger main window.

Figure 3.1 Debugger Window Menu Bar

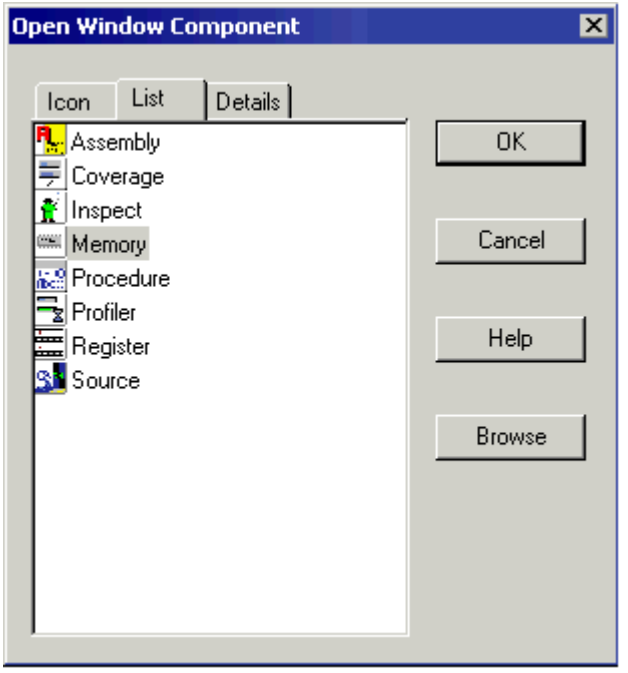


To open the window that lets you choose one or more components:

1. Choose **Component>Open...**
2. In the Open Window Component window shown in Figure 3.2 on page 47, select the desired component.

NOTE To open more than one component, select multiple components.

Figure 3.2 Open Window Component Window



3. In the Open Window Component window, use the mouse to select a component.
4. Click the **OK** button to open the selected component.

There are three tabs in the Open Window Component window:

- The **Icon** tab shows components with large icons.
- The **List** tab shows components with small icons.
- The **Details** tab shows components with their description.

Demo Version Limitations

The demo version limits the number of components you can open at a time. If you use the demo version, you can open a maximum of eight components.

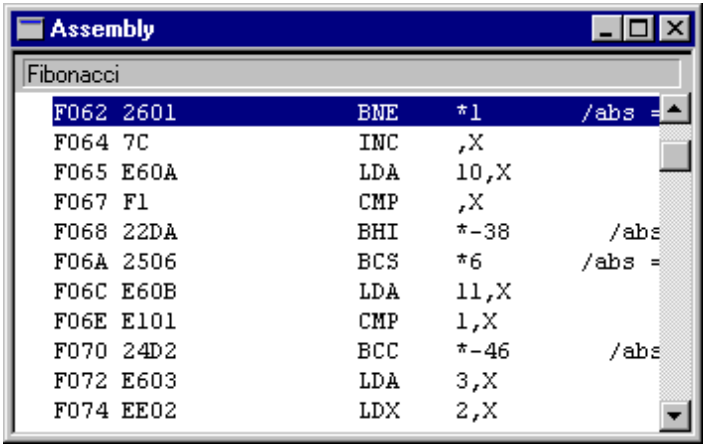
General Debugger Components

This chapter describes the various features and usage of the debugger components.

Assembly Component

The Assembly window, shown in Figure 3.3 on page 48, displays program code in disassembled form. It has a function very similar to that of the Source component window but on a much lower abstraction level. Thus it is therefore possible to view, change, monitor and control the current location of execution in a program.

Figure 3.3 Assembly Window


Assembly
Fibonacci
F062 2601 BNE *1 /abs =
F064 7C INC ,X
F065 E60A LDA 10,X
F067 F1 CMP ,X
F068 22DA BHI *-38 /abs
F06A 2506 BCS *6 /abs =
F06C E60B LDA 11,X
F06E E101 CMP 1,X
F070 24D2 BCC *-46 /abs
F072 E603 LDA 3,X
F074 EE02 LDX 2,X

This window contains all on-line disassembled instructions generated by the loaded application. Each displayed disassembled line in the window can show the following information: the address, machine code, instruction and absolute address in case of a branch instruction. By default, the user can see the instruction and absolute address.

If breakpoints have been set in the application, they are marked in the Assembly component with a special symbol, depending on the kind of breakpoint.

If execution has stopped, the current position is marked in the Assembly component by highlighting the corresponding instruction.

The Object Info Bar of the component window contains the procedure name, which contains the currently selected instruction. When a procedure is double clicked in the Procedure component, the current assembly statement inside this procedure is highlighted in the Assembly component.

Assembly Menu

The Assembly menu shown in Figure 3.4 on page 49 contains all functions associated with the assembly component. Theses entries are described in Table 3.1 on page 49.

Figure 3.4 Assembly Menu

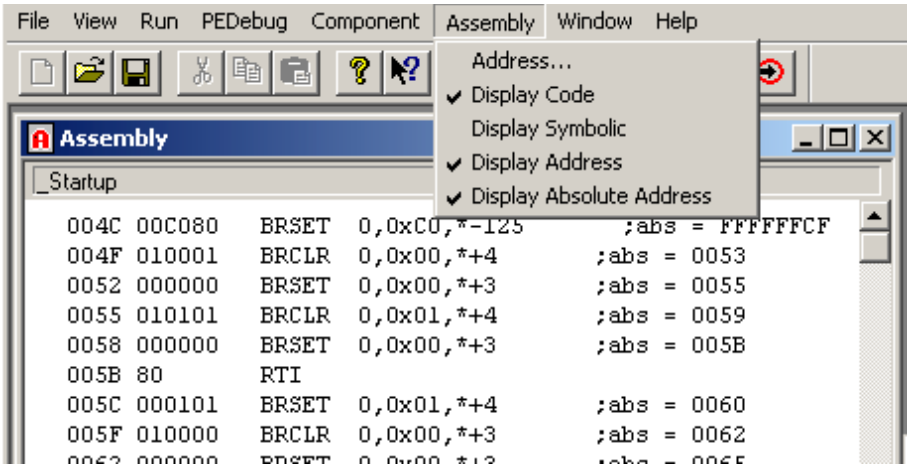


Table 3.1 Assembly Menu Description

Menu Entry	Description
Address...	Opens a dialog box prompting for an address: Show PC.
Display Code	Displays machine code in front of each disassembled instruction.
Display Symbolic	Displays symbolic names of objects.
Display Address	Displays the location address at the beginning of each disassembled instruction.
Display Absolute Address	In a branch instruction, displays the absolute address at the end of the disassembled instruction.

Setting Breakpoints

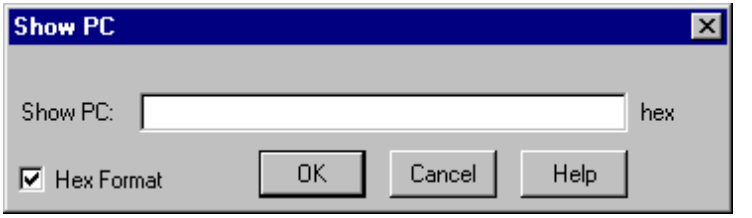
Breakpoints can be set, edited and deleted when using the popup menu. Right-click on any statement in the Source component window, then choose Set Breakpoint, Delete Breakpoint, etc.

NOTE For information on using breakpoints, see Control Points on page 145 chapter.

Show PC Dialog Box

If a hexadecimal address is entered in the Show PC dialog box shown in Figure 3.5 on page 50, memory contents are interpreted and displayed as assembler instructions starting at the specified address.

Figure 3.5 Show PC Dialog Box



Associated Popup Menu

To open the popup menu right-click in the text area of the Assembly component window. The popup menu contains default menu entries for the Assembly component. It also contains some context dependent menu entries described in Table 3.2 on page 51; depending on the current state of the debugger.

Figure 3.6 Assembly Popup Menu

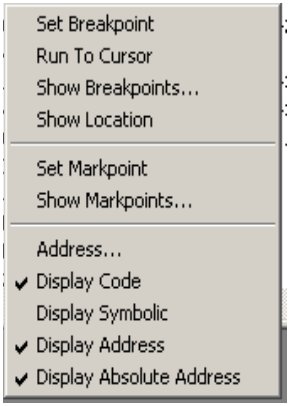


Table 3.2 Assembly Popup Menu Description

Menu Entry	Description
Set Breakpoint	Appears only in the popup menu if no breakpoint is set or disabled on the pointed to instruction. When selected, sets a permanent breakpoint on this instruction. When program execution reaches this instruction, the program is halted and the current program state is displayed in all window components.
Delete Breakpoint	Appears in popup menu if a breakpoint is set or disabled on the specified instruction. When selected, deletes this breakpoint.
Enable Breakpoint	Appears only in popup menu if a breakpoint is disabled on an instruction. When selected, enables this breakpoint.
Disable Breakpoint	Appears in the popup menu if a breakpoint is set on an instruction. When selected, disables this breakpoint.
Run To Cursor	When selected, sets a temporary breakpoint on a specified instruction and continues execution of the program. If there is a disabled breakpoint at this position, the temporary breakpoint will also be disabled and the program will not halt. Temporary breakpoints are automatically removed when they are reached.
Show Breakpoints	Opens the Controlpoints Configuration Window Brealpoints Tab and displays list of breakpoints defined in the application (refer to Control Points on page 145).
Show Location	When selected, highlights the source statement that generated the pointed to assembler instruction. The assembler instruction is also highlighted. The memory range corresponding to this assembler instruction is also highlighted in the memory component.
Set Markpoint	When selected, enables you to set a markpoint at this location.
Delete Markpoint	Appears in the Popup Menu only if a markpoint is set at the nearest code position (visible with marks). When selected, disables this markpoint.
Show Markpoints	Opens the Controlpoints Configuration Window Markpoints Tab and displays list of markpoints defined in the application (refer to Control Points on page 145).
Address...	For a description of the remaining popup menu entries see Table 3.1 “Assembly Menu Description” on page 49.

Retrieving Source Statement

- Point to an instruction in the Assembly component window, drag and drop it into the Source component window. The Source component window scrolls to the source statement generating this assembly instruction and highlights it.
- Left clicking the mouse and clicking the L key Highlights a code range in the Assembly component window corresponding to the first line of code selected in the Source component window where the operation is performed. This line or code range is also highlighted.

Drag Out:

Table 3.3 on page 52 shows the drag actions possible from the Assembly component.

Table 3.3 Assembly Component Drag Actions

Destination Component Window	Action
Command Line	The Command Line component appends the address of the pointed to instruction to the current command.
Memory	Dumps memory starting at the selected instruction PC. The PC location is selected in the memory component.
Register	Loads the destination register with the PC of the selected instruction.
Source	Source component scrolls to the source statements and highlights it.

Drop Into:

Table 3.4 on page 53 shows the drop actions possible in the Assembly component

Table 3.4 Drop Into Assembly Component

Source Component Window	Action
Source	Displays disassembled instructions starting at the first high level language instruction selected. The assembler instructions corresponding to the selected high level language instructions are highlighted in the Assembly component
Memory	Displays disassembled instructions starting at the first address selected. Instructions corresponding to the selected memory area are highlighted in the Assembly component.
Register	Displays disassembled instructions starting at the address stored in the source register. The instruction starting at the address stored in the register is highlighted.
Procedure	The current assembly statement inside the procedure is highlighted in the Assembly component.

Demo Version Limitations

No limitation

Associated Commands

Following commands are associated with the Assembly component:

ATTRIBUTES on page 645, SMEM on page 724, SPC on page 726.

Command Line Component

The Command Line window shown in Figure 3.7 on page 54 interprets and executes all Debugger commands and functions. The command entry always occurs in the last line of the Command component. Characters can be input or pasted on the edit line.

Figure 3.7 Command Line Window

Keying In Commands

You can type Debugger commands after the "**in>**" terminal prompt in the Command Line Component window.

Recalling a Line from the Command Line History

To recall a command in the DOS window use either the up or down arrow,of the F3 funtion key, to retype the previous command.

Scrolling the Command Component Window Content

Use the left and right arrow keys to move the cursor on the line, the HOME key to move the cursor to the beginning of the line, or the END key to move the cursor to the end of the line. To scroll a page, use the PgDn (scroll down a page) or PgUp keys (scroll up a page).

Clearing the Line or a Character of the Command Line

Selected text can be deleted by pressing the left arrow. To clear the current line, press the ESC key.

Command Interpretation

The component executes the command entered, displays results or error messages, if any. Ten previous commands can be recalled using the up arrow key to scroll up or the down arrow key to scroll down. Commands are displayed in blue. Prompts and command responses are displayed in black. Error messages are displayed in red.

When a command is executed and running from the Command Line component, the component cannot be closed. In this case, if the Command Line component is closed with the window close button (X) or with the **Close** entry of the system menu, the following message is displayed:

"Command Component is busy. Closing will be delayed"

The Command Line component is closed as soon as command execution is complete. If the CLOSE on page 665 command is applied to this Command Line component (for

example, from another Command Line component), the component is closed as soon as command execution is finished.

Variable Checking in the Command Line

When specifying a single name as an expression in the command line, this expression is first checked as a local variable in the current procedure. If not found, it is checked as a global variable in the current module. If not found, it is checked as a global variable in the application. If not found, it is checked as a function in the current module. If not found, it is checked as a function in the application, finally if not found an error is generated.

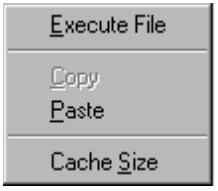
Closing the Command Line during an execution

When a command is executed from a Command Line component, it cannot be closed. If the Command Line component is closed with the close button or with the 'Close' entry of the system menu, the following message is displayed '`Command Component is busy. Closing will be delayed`' and the Command component is closed as soon as command execution is complete. If the 'Close' command is applied to this Command component, the Command component is closed as soon as command execution is complete.

Command Menu

Figure 3.8 on page 55 shows the Command menu, which is identical to the Command Popup menu.

Figure 3.8 Command Menu



Clicking **Execute File** opens a dialog where you can select a file containing Debugger commands to be executed. Theses files generally have a **.cmd** default extension.

Selected text in the Command Line window can be copied to the clipboard by:

- selecting the menu entry **Command>Copy**.
- pressing the CTRL + C key.
- clicking the button in the toolbar.

The **Command>Copy** menu entry and the button are only enabled if something is selected in the Command Line window.

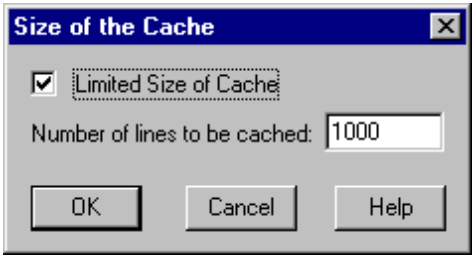
The first line of text contained in the clipboard can be pasted where the caret is blinking (end of current line) by:

- Selecting the menu entry **Command>Paste**
- Pressing CTRL + V simultaneously.
- clicking the icon in the toolbar.

Cache Size

Select **Cache Size** in the menu to bring up the Size of the Cache dialog box and set the cache size in lines for the Command Line window, as shown in Figure 3.9 on page 56.

Figure 3.9 Cache Size Dialog Box



This Cache Size dialog box is the same for the Terminal Component and the TestTerm Component.

Drag Out:

Nothing can be dragged out.

Drop Into:

Memory range, address, and value can be dropped into the Command Line Component window, as described in Table 3.5 on page 57. The command line component appends corresponding items of the current command.

Table 3.5 Drop Into Command Component

Source Component Window	Action
Assembly	The Command Line component appends the address of the pointed to instruction to the current command.
Data	Dragging the name appends the address range of the variable to the current command in the Command Line Window. Dragging the value appends the variable value to the current command in the Command Line Window.
Memory	Appends the selected memory range to the Command Line window
Register	The address stored in the pointed to register is appended to the current command.

Demo Version Limitations

Only 20 commands can be entered and then command component is closed and it is no longer possible to open a new one in the same Debugger session.

Command files with more than 20 commands cannot be executed.

Associated Commands

BD on page 658, CF on page 663, E on page 676, HELP on page 693, NB on page 706, LS on page 703, SREC on page 728, SAVE on page 720.

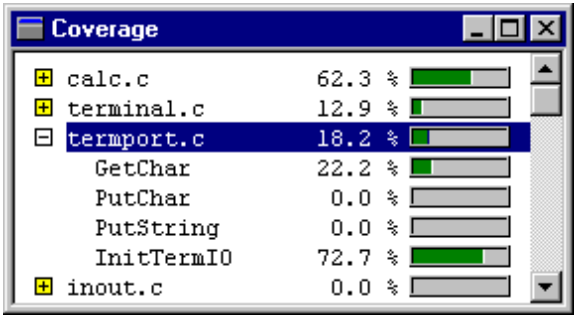
NOTE For more details about commands, refer to Debugger Engine Commands on page 633.

Coverage Component

The Coverage window, shown in Figure 3.10 on page 58 contains source modules and procedure names as well as percentage values representing the proportion of executed code in a given source module or procedure.

Please note that in cases where in cases of advanced code optimizations (like linker overlapping ROM/code areas) the coverage output/data is affected. In such a case, it is recommended to switch of such linker optimizations.

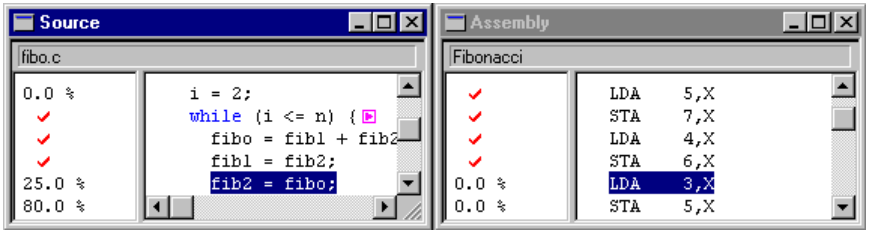
Figure 3.10 Coverage Window



The Coverage window contains percentage numbers and graphic bars. From this component,

You can split views in the Source window and Assembly window, as shown in Figure 3.11 on page 58. A red check mark is displayed in front of each source or assembler instruction that has been executed. Split views are removed when the Coverage window is closed or by selecting Delete in the split view popup menu.

Figure 3.11 Split Views



Coverage Operations

Click the folded/unfolded icons ⊞⊟ to unfold/fold the source module and display/hide the functions defined.

Coverage Menu

The Coverage menu and submenus are shown in Figure 3.12 on page 59.

Figure 3.12 Coverage Menu

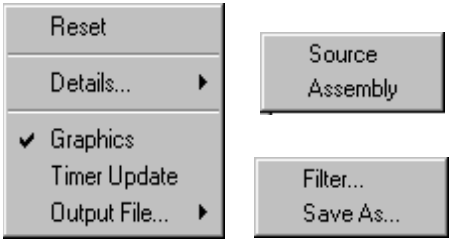


Table 3.6 Coverage Menu Description

Menu Entry	Description
Reset	Resets all simulator statistic information.
Details	Opens a split view in the chosen component (Source or Assembly).
Graphics	Toggles the graphic bars.
Timer Update	Switches the periodic update on/off. If activated, statistics are updated each second.
Output File	Opens the Output File options.

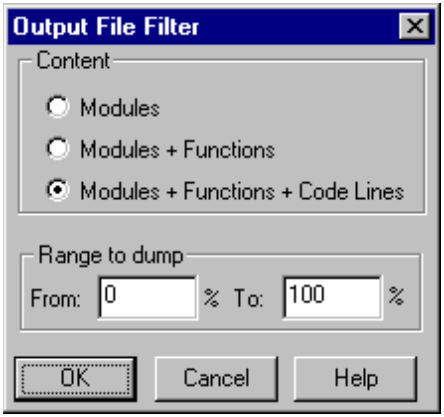
Output File

You can redirect Coverage component results to an output file by selecting **Output File...> Save As...** in the menu or popup menu.

Output File Filter

Select **Output Filter...** to display the dialog box shown in Figure 3.13 on page 60. Select what you want to display, i.e. modules only, modules and functions, or modules, functions and code lines. You can also specify a range of coverage to be logged in your file.

Figure 3.13 Output File Filter Dialog Box



Output File Save

The **Save As...** entry opens a **Save As** dialog where you can specify the output file name and location, an example is shown in Listing 3.1 on page 60.

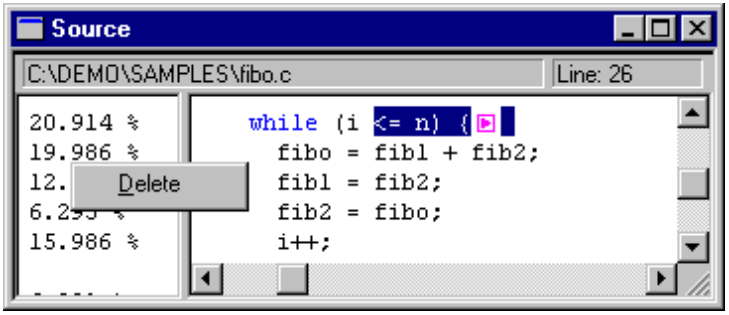
Listing 3.1 Example Output File with Modules and Functions:

```
------------------------------------------------------------
Coverage:       Item:
------------------------------------------------------------
 94.4 %         Application
 FULL           fibo.c
 FULL              Fibonacci()
 FULL              main()
 86.0 %         startup.c
 80.5 %            Init()
 FULL              _Startup()
```

Split View Associated Popup Menu

The popup menu for the split view (Figure 3.14 on page 61) contains the **Delete** entry, which is used to remove the split view.

Figure 3.14 Coverage Split View Associated Popup Menu



Drag Out:

All displayed items can be dragged into a Source or Assembly component. Destination component displays marks in front of the executed source or assembler instruction.

Drop Into:

Nothing can be dropped into the Coverage Component window.

Demo Version Limitations

Only modules are displayed and the Save function is disabled.

Associated Commands

DETAILS on page 673, FILTER on page 682, GRAPHICS on page 693, OUTPUT on page 710, RESET on page 716, TUPDATE on page 734

DA-C Link Component

The DA-C Link window shown in Figure 3.15 on page 62 is an interface module between the DA-C (Development Assistant for C - from RistanCASE GmbH) and the IDE, allowing synchronized debugging features.

Figure 3.15 DA-C Link Window

DA-C Link Operation

When you load the DA-C Link component, communication is established with DA-C (if open) in order to exchange synchronization information.

The **Setup** entry of the DA-C Link main menu allows you to define the connection parameters.

NOTE For related information refer to the Chapter Synchronized Debugging Through DA-C IDE on page 211.

DA-C Link Menu

Selecting Setup from the DA-C Link menu opens the Connection Specification dialog box.

Figure 3.16 DA-C Link Menu

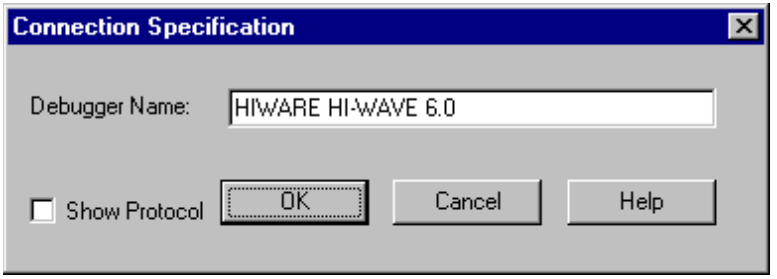
Table 3.7 DA-C Link Menu Description

Menu Entry	Description
Setup	Opens the Connection Specification dialog box.

Connection Specification Dialog Box

In the Connection Specificaton dialog box you can set the DA-C debugger name.

Figure 3.17 Connection Specification Dialog Box



The DA-C debugger name must be the same as the one selected in the DA-C IDE. Check the “Show Protocol" checkbox to display the communication protocol in the Command component of the Debugger. To validate the settings, click the **OK** button. A new connection is established and the "Connection Specification" is saved in the current Project.ini file. The **HELP** button opens the help topic for this dialog.

NOTE If problems exist, refer to the Troubleshooting on page 227 section in the DA-C documentation.

Drag Out

Nothing can be dragged out.

Drop Into

Nothing can be dropped into the DAC Component window.

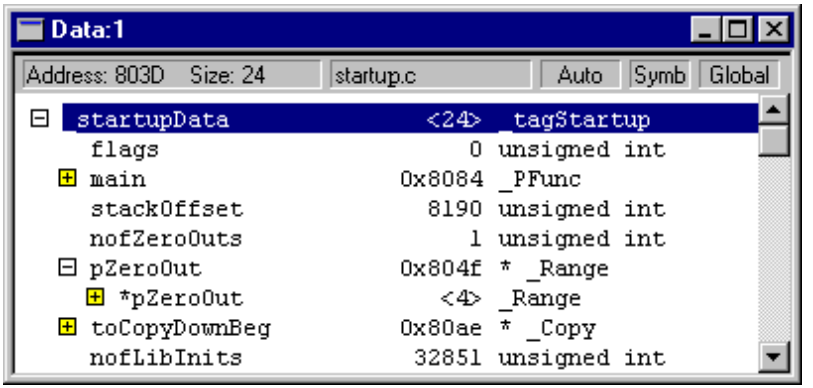
Demo Version Limitations

None.

Data Component

The Data window shown in Figure 3.18 on page 64 contains the names, values and types of global or local variables.

Figure 3.18 Data Window



The Data window shows all variables present in the current source module or procedure. Changed values are in red.

The Component Windows Object Info Bar on page 35 contains the address and size of the selected variable. It also contains the module name or procedure name where the displayed variables are defined, the display mode (automatic, locked, etc.), the display format (symbolic, hex, bin, etc.), and current scope (global, local or user variables).

Various display formats, such as symbolic representation (depending on variable types), and hexadecimal, octal, binary, signed and unsigned formats may be selected.

Structures can be expanded to display their member fields.

Pointers can be traversed to display data they are pointing to.

Watchpoints can be set in this component. Refer to Control Points on page 145 chapter.

Data Operations

- Double-click a variable line to edit the value.
- Click the folded/unfolded icons ⊞⊟ to unfold/fold the structured variable.
- Double-click a blank line: Opens the Expression editor to insert an expression in the Data Component window.
- Select a variable in the Data component, and left mouse button + R key to set a “Read” watchpoint on the selected variable. A green vertical bar is displayed on the

left side of the variables on which a read watchpoint has been defined. If a read access on the variable is detected during execution, the program is halted and the current program state is displayed in all window components.

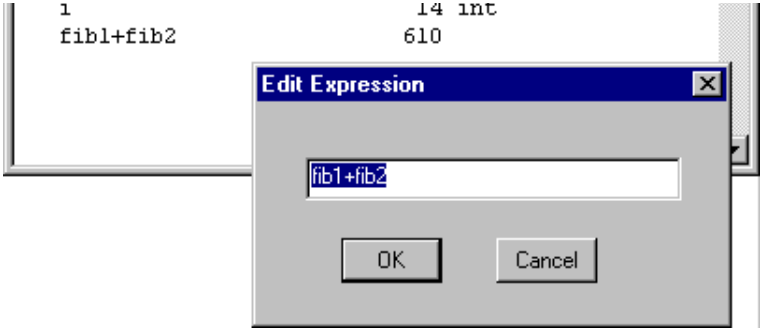
- Select a variable in the Data component, and left mouse button + W key to set a “Write” watchpoint on the selected variable. A red vertical bar is displayed on the left side of the variables on which a write watchpoint has been defined. If write access is detected on the variable during execution, the program is halted and the current program state is displayed in all window components.
- Select a variable in the Data component, and left mouse button + B key to set a “Read/Write” watchpoint on the selected variable. A yellow vertical bar is displayed for the variables on which a read/write watchpoint has been defined. If the variable is accessed during execution, the program is halted and the current program state is displayed in all window components.
- Select a variable on which a watchpoint was previously defined in the Data component, and left mouse button + D key to delete the watchpoint on the selected variable. The vertical bar previously displayed for the variables is removed.
- Select a variable in the Data component, and left mouse button + S key to set a watchpoint on the selected variable. The Watchpoints Setting dialog box is opened. A grey vertical bar is displayed for the variables on which an watchpoint has been defined.

Expression Editor

To add your own expression (in EBNF notation) double-click a blank line in the Data component window to open the **Edit Expression** dialog box shown in Figure 3.19 on page 66, or point to a blank line as shown below and right-click to select **Add Expression...** in the popup menu shown in the figure below.

You may enter a logical or numerical expression in the edit box, using the Ansi-C syntax. In general, this **expression** is a function of one or several variables from the current Data component window.

Figure 3.19 Edit Expression Dialog Box



Example:

With 2 variables **variable_1**, **variable_2**;

expression entered: **(variable_1<<variable_2)+ 0xFF) <= 0x1000** results in a boolean type.

expression entered: **(variable_1>>~variable_2)* 0x1000** will result in an integer type.

NOTE It is not possible to drag an expression defined with the Expression Editor. The "forbidden" cursor is displayed.

Expression Command file

The Expression Command file is automatically generated when a new application is loaded or exiting from the Debugger. User defined expressions are stored in this command file. The name of the expression command file is the name of the application with a .**xpr** extension (.**XPR** file). When loading a new user application, the debugger executes the matching expression command file to load the user defined expression into the data component.

Example: When loading `fibo.abs`, the debugger executes `Fibo.xpr`

Data Menu

Figure 3.20 on page 67 shows the Data component menu, the Zoom submenu is shown in Figure 3.29 on page 72, the Scope submenu is shown in Figure 3.21 on page 68, the Format submenu in Figure 3.22 on page 68, the Mode submenu in Figure 3.24 on page 70, the Options submenu in Figure 3.26 on page 71 and the Zoom and Sort submenus in Figure 3.29 on page 72. Data Menu entries are described in Table 3.8 on page 67.

Figure 3.20 Data Menu

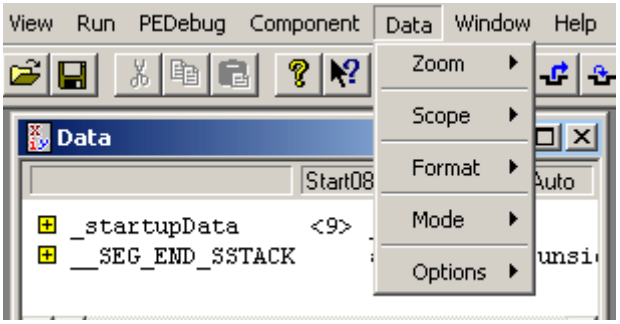


Table 3.8 Data Menu Entry Description

Menu Entry	Description
Zoom	Zooms in or out of the selected structure. The member field of the structure replaces the variable list.
Scope...	Opens a variable display submenu.
Format...	Symb, Hex (hexadecimal), Oct (octal), Bin (binary), Dec (signed decimal), UDec (unsigned decimal) display format.
Mode...	Switches between Automatic, Periodical, Locked, and Frozen update mode.
Options...	Opens an options menu for data, for example, Pointer as Array facility.

Scope Submenu

The Scope Submenu is activated by highlighting the Scope entry on the Data menu:

Figure 3.21 Scope Submenu

Table 3.9 on page 68 describes the Scope submenu entries.

Table 3.9 Scope Submenu Entries

Menu Entry	Description
Global	Switches to **Global** variable display in the Data component.
Local	Switches to **Local** variable display in the Data component.
User	Switches to **User** variable display in the Data component. Displays user defined expression (variables are erased).

NOTE If the data component mode is not automatic, entries are greyed (because it is not allowed to change the scope).

In Local Scope, if the Data component is in Locked or Periodical mode, values of the displayed local variables could be invalid (since these variables are no longer defined in the stack).

Format Submenu

The Format Submenu is activated by highlighting the format entry on the Data menu:

Figure 3.22 Format Submenu

Table 3.10 on page 68 describes the Format submenu entries.

Table 3.10 Format Sub Menu Entries

Menu Entry	Description
Selected	The changes will be applied to the selection only
All	The changes will be applied to all items

Format Selected & All Sub Menu

The Format Selected & All Submenu is activated by highlighting this entry on the Data Component menu:

Figure 3.23 Format Selected & All Submenus

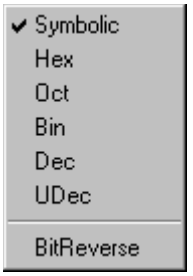


Table 3.11 on page 69 describes the Format Selected Mode & Format All Mode Sub Menu entries.

Table 3.11 Format Selected & All Sub Menu

Menu entry	Description
Symbolic	Select the **Symbolic** (display format depends on the variable type) display format. This is the default display.
Hex	Select the hexadecimal data display format
Bin	Select the binary data display format
Oct	Select the octal data display format
Dec	Select the signed decimal data display format
UDec	Select the unsigned decimal data display format
Bit Reverse	Select the bit reverse data display format (Each bit is reversed).

Mode Submenu

The Mode Submenu is activated by highlighting the **Mode** entry on the Data menu:

Figure 3.24 Mode Submenu

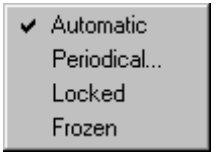


Table 3.12 on page 70 describes the Mode Submenu entries.

Table 3.12 Mode Submenu

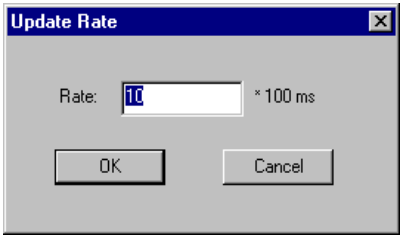
Menu Entry	Description
Automatic	Switches to **Automatic** mode (default), variables are updated when the connection is stopped. Variables from the currently executed module or procedure are displayed in the data component.
Periodical	Switches to **Periodical** mode: variables are updated at regular time intervals when the connection is running. The default update rate is 1 second, but can be modified by steps of up to 100 ms using the associated dialog box (see below).
Locked	Switches to **Locked** mode, value from variables displayed in the data component are updated when the connection is stopped.
Frozen	Switches to **Frozen** mode: value from variables displayed in the data component are not updated when the connection is stopped.

NOTE In Locked and Frozen mode, variables from a specific module are displayed in the data component. The same variables are always displayed in the data component.

Update Rate Dialog Box

The Update Rate dialog box shown in Figure 3.25 on page 71 allows you to modify the default update rate using steps of 100 ms.

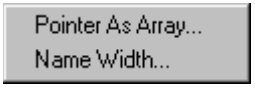
Figure 3.25 Update Rate Dialog Box



Options Submenu

The Options Submenu is activated by highlighting the Options entry on the Data menu:

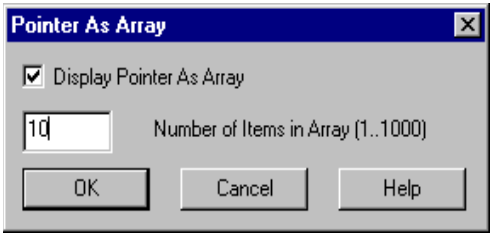
Figure 3.26 Options Submenu



Pointer as Array Option

In the Data menu's Options submenu, choose **Options...>Pointer as Array...** to open the dialog box shown in Figure 3.27 on page 71.

Figure 3.27 Pointer as Array Dialog Box

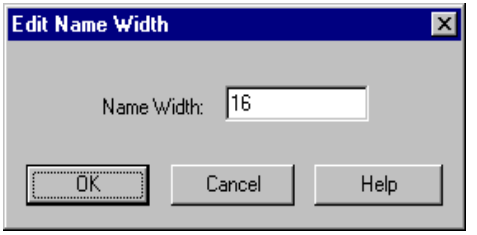


Within this dialog box, you can display pointers as arrays, assuming that the pointer points to the first item (**pointer[0]**). Note that this setup is valid for all pointers displayed in the Data window. Check the **Display Pointer as Array** checkbox and set the number of items that you want to be displayed as array items.

Name Width Option

In the Data Menu's Options submenu, choose **Options... > Name Width...** to open the dialog box shown in Figure 3.28 on page 72.

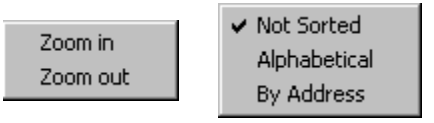
Figure 3.28 Edit Name Width Dialog Box



This dialog box allows you to adjust the width of the variable name displayed in the Data window. This string will be cut off if it is longer than 16 characters. Thus, by enlarging the value you can adapt the window to longer names.

Zoom and Sort Submenus

Figure 3.29 Zoom and Sort Submenus



Associated Popup Menu

Figure 3.30 Data Popup Menu

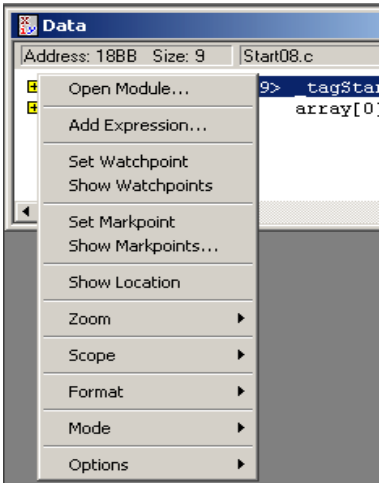


Table 3.13 on page 73 specifies the Data Popup Menu entries.

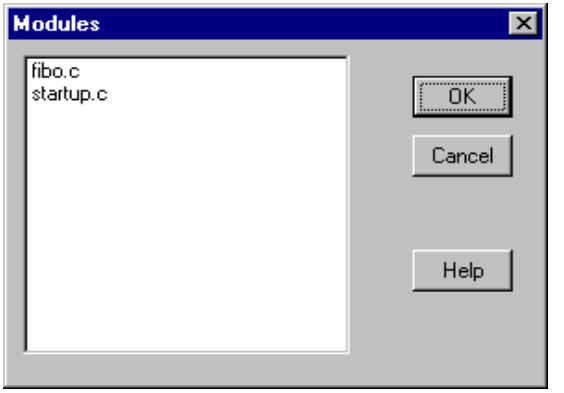
Table 3.13 Data Popup Menu

Menu Entry	Description
Open Module...	Opens the Open Module dialog box.
Set Watchpoint	Appears only in the popup menu if no watchpoint is set or disabled on the pointed to variable. When selected, sets a read/write watchpoint on this variable. A yellow vertical bar is displayed for the variables on which a read/write watchpoint has been defined. If the variable is accessed during execution, the program is halted and the current program state is displayed in all window components.
Delete Watchpoint	Appears only in the popup menu if a watchpoint is set or disabled on the pointed to variable. When selected, deletes this watchpoint.
Enable Watchpoint	Appears only in the popup menu if a watchpoint is disabled on the pointed to variable. When selected, enables this watchpoint.
Disable Breakpoint	Appears only in the popup menu if a breakpoint is set on the pointed to instruction. When selected, disables this watchpoint.
Show Watchpoints	Opens the Watchpoints Setting dialog box and allows you to view the list of watchpoints defined in the application. (Refer to Control Points on page 145).
Show location	Forces all open components to display information about the pointed to variable (e.g., the Memory component selects the memory range where the variable is located).

SUBMENU Open Module

The dialog shown in Figure 3.31 on page 74 lists all source files bound to the application. Global variables from the selected module are displayed in the data component. This is only supported when the component is in **Global** scope mode.

Figure 3.31 Open Modules Dialog Box



Drag Out:

Table 3.14 on page 74 describes the drag actions possible from the Data component.

Table 3.14 Dragging Data Possibilities

Destination Component Window	Action
Command Line	Dragging the name appends the address of the variable to the current command in the Command Line Window. Dragging the value appends the variable value to the current command in the Command Line Window.
Memory	Dumps memory starting at the address where the selected variable is located. The memory area where the variable is located is selected in the memory component.
Source	Dragging the name of a global variable in the source Window displays the module where the variable is defined and first occurrence of the variable is highlighted.
Register	Dragging the name loads the destination register with the address of the selected variable. Dragging the value loads the destination register with the value of the variable.

NOTE It is important to distinguish between dragging a variable name and dragging a variable value. Both operations are possible. Dragging the name drags the address of the variable. Dragging the variable value drags the value.

NOTE Expressions are evaluated at run time. They do not have a location address, so you cannot drag an expression name into another component. Values of expressions can be dragged to other components.

Drop Into:

Table 3.15 on page 75 describes the drop actions possible in the Data component.

Table 3.15 Data Drop Possibilities

Source Component Window	Action
Source	A selection in the Source window is considered an expression in the Data window, as if it was entered through the Expression Editor of the Data component. Refer to Data Component on page 64, Expression Editor on page 65.
Module	Displays the global variables from the selected module in the data component.

Demo Version Limitations

Only 2 variables can be displayed.

Only 2 members of a structure are visible when unfolded.

Only 1 expression can be defined.

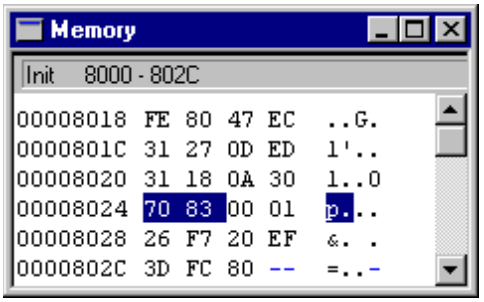
Associated Commands

ADDXPR on page 645, ATTRIBUTES on page 645, DUMP on page 675, PTRARRAY on page 713, SMOD on page 725, SPROC on page 727, UPDATERATE on page 738, ZOOM on page 744.

Memory Component

The Memory window shown in Figure 3.32 on page 76 displays unstructured memory content or memory dump, i.e. continuous memory words without distinction between variables.

Figure 3.32 Memory Window



Various data formats (byte, word, double) and data displays (hexadecimal, binary, octal, decimal, unsigned decimal) can be specified for the display and edition of memory content.

Watchpoints can be defined in this component.

NOTE Refer to Control Points on page 145 for more information about watchpoints.

Memory areas can be initialized with a fill pattern using the Fill Memory on page 81 box.

An ASCII dump can be added/removed on the right side of the numerical dump when checking/unchecking **ASCII** in the **Display** menu entry.

The location address may also be added/removed on the left side of the numerical dump when checking/unchecking **Address** in the **Display** menu entry.

To specify the start address for the memory dump use the **Address** menu entry.

The Component Windows Object Info Bar on page 35 contains the procedure or variable name, structure field and memory range matching the first selected memory word.

"uu" memory value means: not initialized.

"--" memory values mean: not configured (no memory available)

NOTE Memory values that have changed since the last refresh status are displayed in red. However, if a memory item is edited or rewritten with the same value, the display for this memory item remains black.

Memory Operations

- Double-click a memory position to edit it. If the memory is not initialized, this operation is not possible.
- Drag the mouse in the memory dump to select a memory range.
- Hold down the left mouse button + A key to jump to a memory address. The pointed to value is interpreted as an address and the memory component dumps memory starting at this address.
- Select a memory range, and hold down the left mouse button + R key to set a “Read” watchpoint for the selected memory area. Memory ranges where a read watchpoint has been defined are underlined in green. If read access on the memory area is detected during execution, the program is halted and the current program state is displayed in all window components.
- Select a memory range, and hold down the left mouse button + W key to set a “Write” watchpoint on the selected memory area. Memory ranges where a write watchpoint has been defined are underlined in red. If write access on the memory area is detected during execution, the program is halted and the current program state is displayed in all window components.
- Select a memory range, and hold down the left mouse button + B key to set a “Read/Write” watchpoint on the selected memory area. Memory ranges where a read/write watchpoint has been defined are underlined in black. If the memory area is exceeded during execution, the program is halted and the current program state is displayed in all window components.
- Select a memory range on which a watchpoint was previously defined, and hold down the left mouse button + D key to delete the watchpoint on the selected memory area. The memory area is no longer underlined.
- Select a memory range, and hold down the left mouse button + S key to set a watchpoint on the selected memory area. The Watchpoints Setting dialog box is opened. Memory ranges where a watchpoint has been defined are underlined in black.

Memory Menu

The Memory Menu shown in Figure 3.33 on page 78 provides access to memory commands. Table 3.16 on page 78 describes the menu entries.

Figure 3.33 Memory Menu

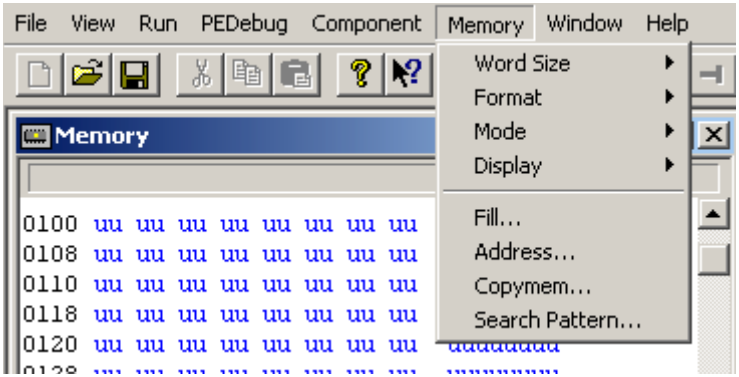


Table 3.16 Memory Menu Description

Menu Entry	Description
Word size	Opens a submenu to specify the display unit size.
Format	Opens a submenu to select the format to display items.
Mode	Opens a submenu to choose the update mode.
Display	Opens a submenu to toggle the display of addresses and ASCII dump.
Fill...	Opens the Fill Memory on page 81 to fill a memory range with a bit pattern.
Address...	Opens the memory dialog and prompts for an address.
CopyMem	Opens the CopyMem dialog box that allows you to copy memory range values to a specific location.
Search Pattern	Opens the Search Pattern dialog box.

Word Size Submenu

With the Word Size submenu shown in Figure 3.34 on page 79, you can set the memory display unit. Table 3.17 on page 79 describes the menu entries.

Figure 3.34 Word Size Submenu

Table 3.17 Word Size Submenu Description

Menu Entry	Description
Byte	Sets display unit to byte size.
Word	Sets display unit to word size (=2 bytes).
Lword	Sets display unit to Lword size (=4 bytes).

Format Submenu

With the Format Submenu shown in Figure 3.35 on page 79, you can set the memory display format. Table 3.18 on page 79 describes the menu entries.

Figure 3.35 Format Submenu

Table 3.18 Format Submenu Description

Menu Entry	Description
Hex	Selects the hexadecimal memory display format
Bin	Selects the binary memory display format
Oct	Selects the octal memory display format
Dec	Selects the signed decimal memory display format

Table 3.18 Format Submenu Description (*continued*)

Menu Entry	Description
UDec	Selects the unsigned decimal memory display format
Bit Reverse	Selects the bit reverse memory display format (each bit is reversed).

Mode Submenu

With the Mode submenu shown in Figure 3.36 on page 80, you can set the memory mode format. Table 3.19 on page 80 describes the menu entries.

Figure 3.36 Mode Submenu

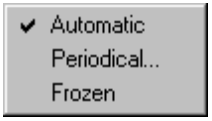


Table 3.19 Mode Submenu Description

Menu Entry	Description
Automatic	Selects **Automatic** mode (default), memory dump is updated when the connection is stopped.
Periodical	Selects the **Periodical** mode, memory dump is updated at regular time intervals when the connection is running. The default update rate is 1 second, but it can be modified by steps of up to 100 ms using the associated dialog box (see below).
Frozen	Selects the **Frozen** mode, memory dump displayed in the memory component is not updated when the connection is stopped.

Display Submenu

With the Display submenu shown in Figure 3.37 on page 80, you can set the memory display (address/ascii). Table 3.20 on page 81 describes the menu entries.

Figure 3.37 Display Submenu

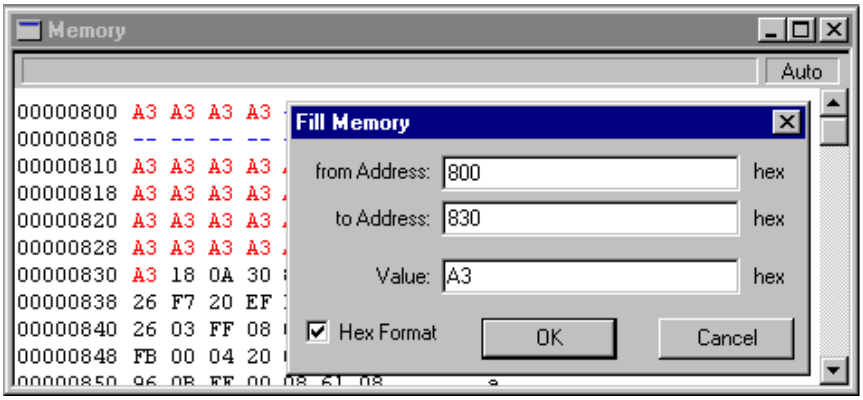
Table 3.20 Display Submenu Description

Menu Entry	Description
Address	Allows you to toggle the display of address dump.
ASCII	Allows you to toggle the display of ASCII dump.

Fill Memory

The Fill Memory dialog box shown in Figure 3.38 on page 81 allows you to fill a memory range (**from Address** edit box and **to Address** edit box) with a bit pattern (**value** edit box).

Figure 3.38 Fill Memory Dialog Box

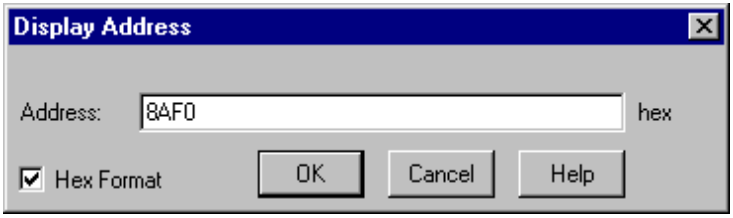


NOTE If “Hex Format” is checked, numbers and letters are interpreted as hexadecimal numbers. Otherwise, expressions can be typed and Hex numbers should be prefixed with “**Ox**” or “**$**”.

Display Address

With the Display Address dialog box, shown in Figure 3.39 on page 82,, the memory component dumps memory starting at the specified address.

Figure 3.39 Display Address Dialog Box

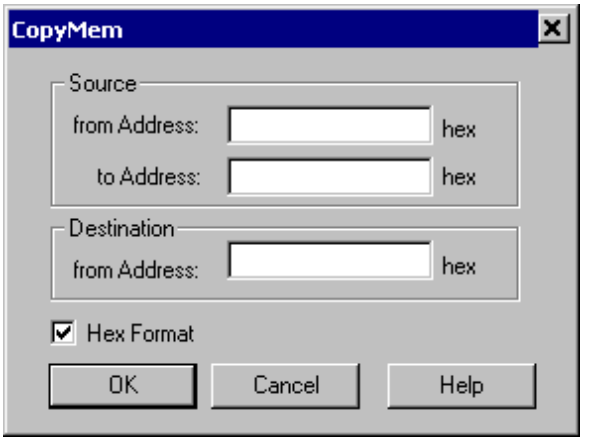


NOTE The **Show PC** dialog box is the same as the Display Address dialog box. In this dialog box, the Assembly component dumps assembly code starting at the specified address.

CopyMem Submenu

The CopyMem dialog box shown in Figure 3.40 on page 83 allows you to copy a memory range to a specific address.

Figure 3.40 CopyMem Dialog Box



To copy a memory range to a specific address, enter the source range and the destination address. Press the `OK` button to copy the specified memory range. Press the `Cancel` button to close the dialog without changes. Press the `Help` button to open the help file associated with this dialog.

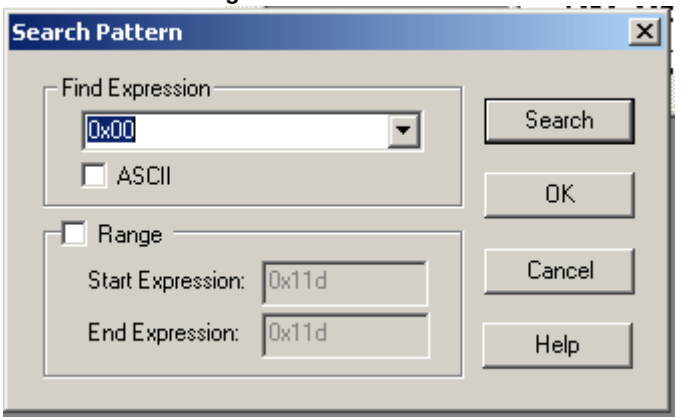
If "**Hex Format**" is checked, all given values are in Hexadecimal Format. You don't need to add "0x". For instance type 1000 instead of 0x1000.

NOTE If you try to read or write to an unauthorized memory address, an error dialog box appears.

Search Pattern

The Search Pattern dialog box shown in Figure 3.41 on page 84 allows you to search memory or a memory range for a specific expression.

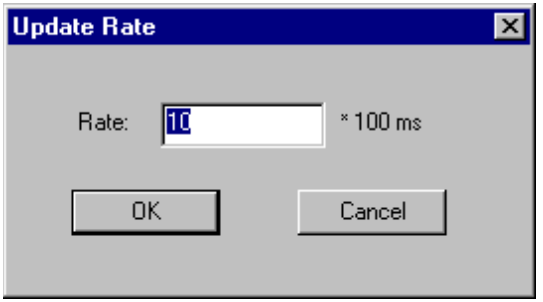
Figure 3.41 Search Pattern Dialog Box



Update Rate

This dialog box shown in Figure 3.42 on page 84 allows you to modify the update rate in steps of 100ms.

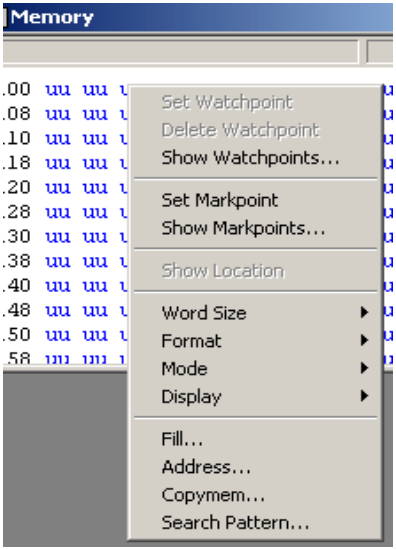
Figure 3.42 Update Rate Dialog Box



NOTE Periodical mode is not available for all hardware connections or some additional configuration may be required in order to make it work.

Associated Popup Menu

Figure 3.43 Memory Popup Menu



The Memory popup menu entries shown in Table 3.21 on page 85 allow you to execute memory associated commands.

Table 3.21 Memory Popup Menu Description

Menu Entry	Description
Set Watchpoint	Appears in the Popup Menu only if no watchpoint is set or disabled on the selected memory range. When selected, sets a Read/Write watchpoint at this memory area. Memory ranges where a read/write watchpoint has been defined are underlined in yellow. If the memory area is accessed during execution of the application, the program is halted and the current program state is displayed in all window components.
Delete Watchpoint	Appears in the Popup Menu only if a watchpoint is set or disabled on the selected memory range. When selected, deletes this watchpoint.
Show Watchpoints	When selected, brings up the Controlpoints Configuration Window - Watchpoints Tab. This is the interface through which watchpoints are controlled. (See "Control Points" chapter)

Table 3.21 Memory Popup Menu Description (*continued*)

Menu Entry	Description
Set Markpoint	Appears in the Popup Menu only if no watchpoint is set or disabled on the selected memory range. When selected, sets a Read/Write watchpoint at this memory area.
Show Markpoints	When selected, brings up the Controlpoints Configuration Window - Markpoints Tab. This is the interface through which markpoints are controlled. (See "Control Points" chapter)
Show Location	Forces all opened windows to display information about the selected memory area.
Word Size, etc.	The remaining entries in this menu are explained in table 3.17 Memory Menu Description on page 78

Drag Out:

Table 3.22 on page 86 Describes the drag actions possible from the Memory component.

Table 3.22 Memory Component Drag Possibilities

Destination Component Window	Action
Assembly	Displays disassembled instructions starting at the first address selected. The instructions corresponding to the selected memory area are highlighted in the Assembly component.
Command Line	Appends the selected memory range to the Command Line window
Register	Loads the destination register with the start address of the selected memory block.
Source	Displays high level language source code starting at the first address selected. Instructions corresponding to the selected memory area are greyed in the source component.

Drop Into:

Table 3.23 on page 87 shows the drop actions possible in the Memory component.

Table 3.23 Memory Component Drop Possibilities

Source Component Window	Action
Assembly	Dumps memory starting at the selected PC instruction. The PC location is selected in the memory component.
Data	Dumps memory starting at the address where the selected variable is located. The memory area where the variable is located is selected in the memory component.
Register	Dumps memory starting at the address stored in the selected register. The corresponding address is selected in the memory component.
Module	Dumps memory starting at the address of the first global variable in the module. The memory area where this variable is located is selected in the memory component.

Demo Version Limitations

No limitation

Associated Commands

ATTRIBUTES on page 645, FILL on page 681, SMEM on page 724, SMOD on page 725, SPC on page 726, UPDATERATE on page 738.

MicroC Component

The MicroC window shown in Figure 3.44 on page 88 is an interface module for RHAPSODY in MicroC, the analysis, design and implementation tool for embedded systems and software developers from I-LOGIX.

Figure 3.44 MicroC Window

The MicroC component establishes a communication with Rhapsody in MicroC to activate its design-level debugging capabilities. Rhapsody in MicroC drives its debugging animation that communicates with the Debugger environment over TCP/IP. This allows you to execute, stop and run the application, to set step commands, breakpoints, events, and idle states to perform control over the application.

Communication is realized by selecting the **Connect** entries of the **MicroC Link** menu. The **Setup** entry allows you to define the connection parameters.

The functions available allow you to start the currently loaded application, to stop it, to execute a single step in the application, to set and clear a breakpoint, to evaluate an expression and to quit the application interface.

NOTE For more information, refer to the RHAPSODY in MicroC documentation from I-Logix.

NOTE In order to work, MicroC needs to have a copy of the `amc_communication_dll.dll` in the `prog` directory from the current installation.

MicroC Link Menu

Figure 3.45 on page 88 shows the MicroC menu and its entries are described in Table 3.24 on page 89.

Figure 3.45 MicroC Link Menu

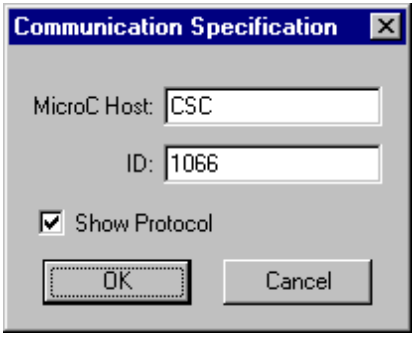
Table 3.24 MicroC Link Menu Description

Menu Entry	Description
Setup	Opens the communication setup Window.
Connect	Establishes communication with RHAPSODY in MicroC.

MicroC Communication Specification

Within the Communication Specification dialog box shown in Figure 3.46 on page 89, you can set the MicroC Host and ID for communication between the Debugger and RHAPSODY in MicroC. A checkbox allows you to see the communication protocol.

Figure 3.46 Communication Specification



Drag Out:

Nothing can be dragged out.

Drop Into:

Nothing can be dropped in.

Demo Version Limitations

The MicroC Component is not available in demo mode.

MicroC DLLs

The RiMC (or MicroC.wnd) component has been updated to make use of the new features that come of the latest release of the communication DLL from I-Logix.

To ensure proper communication between Rhapsody in MicroC and the external debugger (HI-WAVE) from Freescale (formerly HIWARE), two files have to be installed in the 'prog' subdirectory of the CodeWarrior installation:

microc.wnd

This is the HI-WAVE component that has to be loaded in order to configure the communication parameters and mode of operation. This component requires the amc_communication_dll.dll to be loaded properly (if this DLL is missing, there will be an error message that a library is missing).

amc_communication_dll.dll

This DLL implements the actual protocol (over TCP/IP). This DLL is delivered together with the RiMC and has to be copied into the 'prog' subdirectory of the CodeWarrior installation (this DLL will not be installed with the CodeWarrior product).

The 'Product Version' of this DLL has to be 'RiMC 3.0' of higher.

Changes and New Features

The new DLL from I-Logix allows now implementing the Graphical Back Animation with fewer resources on the target system; so only one single breakpoint is required in synchronous mode and even none in asynchronous mode!

- There are now two modes of operation:

  Synchronous:

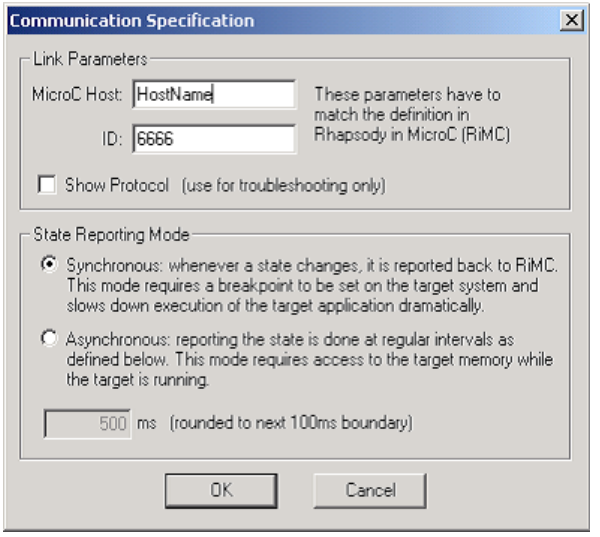
  This mode corresponds to the legacy implementation and lets RiMC update the state whenever a change of state is detected on the target system. This is implemented by setting a breakpoint on the target on a function that is called whenever that state of the application is changed. When hit, the state is sent to RiMC and the application is resumed immediately. By concept, this procedure will slow down execution of the target application dramatically. Compared to the previous releases, only one single breakpoint is required for this mode.

  Asynchronous

  This is a new mode introduced in this release. The state of the application will only sampled from time to time. Thus, this mode allows the application to run at full speed but will not update RiMC about each change of state. Also, it does not require any resources on the target system except that the target memory has to be accessible while the application is running. The connections that support this mode are the HC(S)12(X) Freescale Full Chip Simulator and any Host connection (HTI) that uses the BDM of features dual-ported RAM.

- The Setup dialog was extended to reflect that additional modes:

Figure 3.47 Communication Specification


Communication Specification
Link Parameters
MicroC Host:
HostName
ID:
6666
These parameters have to match the definition in Rhapsody in MicroC (RiMC)
Show Protocol (use for troubleshooting only)
State Reporting Mode
Synchronous: whenever a state changes, it is reported back to RiMC. This mode requires a breakpoint to be set on the target system and slows down execution of the target application dramatically.
Asynchronous: reporting the state is done at regular intervals as defined below. This mode requires access to the target memory while the target is running.
500
ms (rounded to next 100ms boundary)
OK
Cancel

In **Asynchronous mode**: the interval for updating the state can be specified in increments of 100ms. All the settings from this dialog are saved in the current project file and will be used in future sessions automatically.

- There are now command line commands to setup the communication parameters:

`MCPROTOCOL [ON|OFF]`

Switched on and off the protocol to the Command window (when open at all).

`MCMODE (SYNC|ASYNC [interval])`

Sets the reporting mode to synchronous or asynchronous. If asynchronous is specified, the interval can be specified too. If the interval is not specified, the previous value will be maintained.

`MCCONNECT [HostName] [portNumber]`

This command tries to connect to RiMC. The name of the computer where RiMC is expected and/or its port number can be can be specified. If not specified, the previous value will be used.

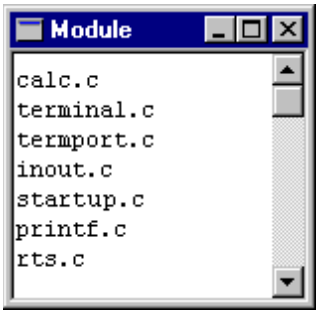
Each of these commands will close any pending communication and re-establish communication with the new parameters.

In Synchronous mode: the states are reported not faster than every 10ms. This will avoid overruns in the communication to RiMC when using the simulator as a connection.

Module Component

The Module window shown in Figure 3.48 on page 92 gives an overview of source modules building the application.

Figure 3.48 Module Window



The Module component displays all source files (source modules) bound to the application. The Module window displays all modules in the order they appear in the absolute file.

Module Operations

Double-clicking a module name forces all open windows to display information about the module: the Source Component window shows the module's source and the global Data Component window displays the module's global variables.

Module Menu

The Module Component window has no menu.

Drag Out:

Table 3.25 on page 93 shows the drag actions possible from the Module component.

Table 3.25 Module Component Drag Possibllities

Destination Component Window	Action
Data > Global	Displays the global variables from the selected module in the data component
Memory	Dumps memory starting at the address of the first global variable in the module. The memory area where this variable is located is selected in the memory component.
Source	Displays the source code from the selected module.

Drop Into:

Nothing can be dropped into the Module Component window.

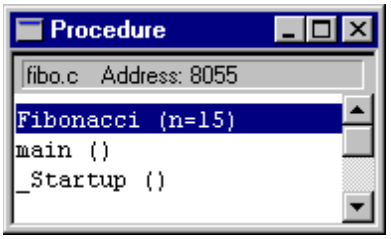
Demo Version Limitations

Only 2 modules are displayed

Procedure Component

The Procedure window shown in Figure 3.49 on page 94 displays the list of procedure or function calls that have been made up to the moment the program was halted. This list is known as the ‘procedure chain’or the ‘call chain’.

Figure 3.49 Procedure Window



In the Procedure Component window, entries in the call chain are displayed in reverse order from the last (most recent on top) call to the first call (initial on bottom). Types of procedure parameters are also displayed.

The Object Info bar of the component window contains the source module and address of the selected procedure.

Procedure Operations

Double-clicking on a procedure name forces all open windows to display information about that procedure: the Source Component window shows the procedure's source, the local Data Component window displays the local variables and parameters of the selected procedure. The current assembly statement inside this procedure is highlighted in the Assembly component.

NOTE When a procedure of a level greater than 0 (the top most) is double clicked in the Procedure Component, the statement corresponding to the call of the lower procedure is selected in the Source Window and Assembly Window.

Procedure Menu

Figure 3.50 on page 95 shows the Procedure menu and its entries are described in Table 3.26 on page 95.

Figure 3.50 Procedure Menu

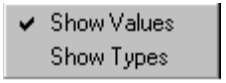


Table 3.26 Procedure Menu Description

Menu Entry	Description
Show Values	Switches to the display of function parameter values in the procedure component.
Show Types	Toggles to the display of function parameter types in the procedure component.

Drag Out:

Table 3.27 on page 95 shows the drag actions possible from the Procedure component.

Table 3.27 Procedure Component Drag Possibilities .

Destination Component Window	Action
Data > Local	Displays the local variables from the selected procedure in the data component
Source	Displays source code of the selected procedure. Current instruction inside the procedure is highlighted in the Source component.
Assembly	The current assembly statement inside the procedure is highlighted in the Assembly component.

Drop Into:

Nothing can be dropped into the Procedure component.

Demo Version Limitations

Only the last two procedures are displayed.

Associated Commands

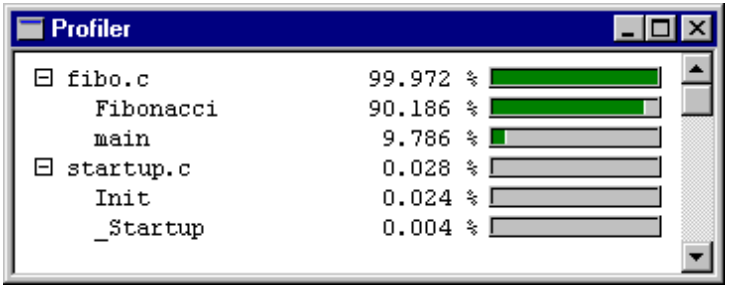
ATTRIBUTES on page 645, FINDPROC on page 684

Profiler Component

The Profiler window shown in Figure 3.51 on page 96 provides information on application profile.

NOTE In cases where in cases of advanced code optimizations (like linker overlapping ROM/code areas) the profiler output/data is affected. In such a case, it is recommended to switch of such linker optimizations.

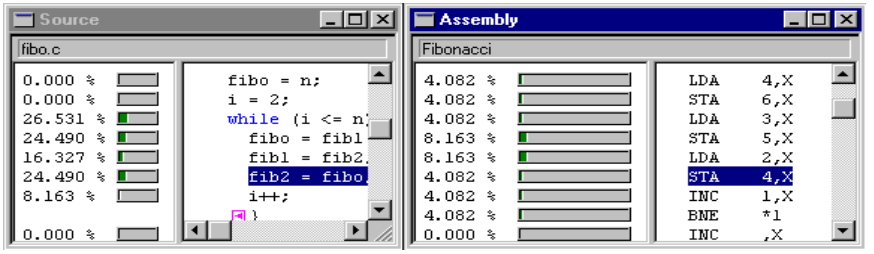
Figure 3.51 Profiler Window



The Profiler window contains source module and procedure names and percentage values representing the time spent in each source module or procedure. The Profiler component window contains percentages and also graphic bars.

The Profiler window can set a split view in the Source and Assembly windows, as shown in Figure 3.52 on page 96. To obttain a split view in either the Source or Assembly windows, select: **Details>Source** or **Details>Assembly** or both from the Profilier menu and submenu. The split windows effect ends when the Profilier window is closed.

Figure 3.52 Split View in the Source and Assembly Windows



Percentage values representing the time spent in each source or assembler instruction are displayed on the left side of the instruction. The split view can also display graphic bars.

Split views are removed when the Coverage component is closed or if you open the split view Popup Menu and select **Delete**.

The value displayed may reflect percentages from total code or percentages from module code.

Profiler Operations

Click the fold/unfold icon to unfold/fold the source module.

Profiler Menu

Figure 3.53 on page 97 shows the Profiler Menu entries, with the Details submenu and the Base submenu. Figure 3.54 on page 97 shows the Profiler Output File submenu. Entries are described in Table 3.28 on page 97.

Figure 3.53 Profiler Menu and Submenus

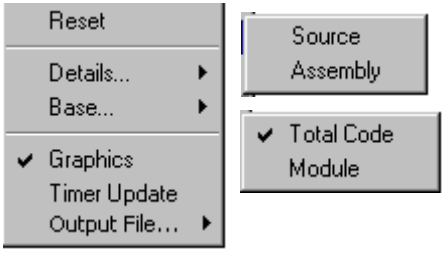


Figure 3.54 Profiler Output File Submenu

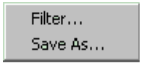


Table 3.28 Profiler Menu Entries Description

Menu Entry	Description
Reset	Resets all statistics.
Details	Sets a split view in the chosen component (Source or Assembly)
Base	Sets the base of percentage (total code or module code).
Graphics	Toggles the display from graphics bar.

Table 3.28 Profiler Menu Entries Description (*continued*)

Menu Entry	Description
Timer Update	Switches on/off the periodic update of the Coverage component. If activated, statistics are updated each second.
Output File	Setup the Profiler Output File Functions on page 98.

Split View Associated Popup Menu

Figure 3.55 on page 98 shows the Profiler popup menu, the **Delete** and **Graphics** menu entries are described in Table 3.29 on page 98.

Figure 3.55 Profiler Split View Associated Popup Menu

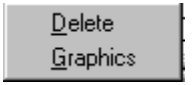


Table 3.29 Profiler Split View Associated Popup Menu Description

Menu Entry	Description
Delete	Removes the split view from the host component.
Graphics	Toggles the graphic bars display in the split view.

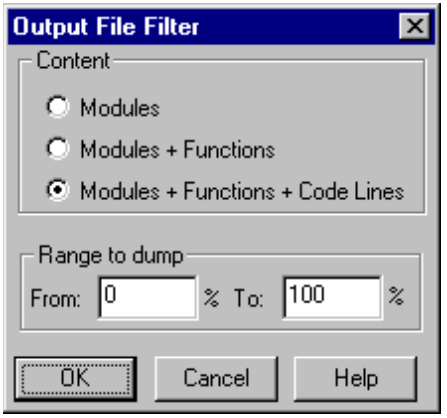
Profiler Output File Functions

You can redirect the Profiler component results to an output file by choosing **Output File...> Save As...** in the menu or popup menu.

Output File Filter

By choosing **Output Filter...**, the dialog box shown in Figure 3.56 on page 99 lets you select what you want to display, i.e. modules only, modules and functions, or modules and functions and code lines. You can also specify a range of coverage to be logged in your file.

Figure 3.56 Output File Filter Dialog Box



Output File Save

The **Save As...** entry opens a **Save As** dialog box where you can specify the output file name and location.

Associated Popup Menu

Identical to menu.

Drag Out:

All displayed items can be dragged out. Destination windows may display information about the time spent in some codes in a split view.

Drop Into:

Nothing can be dropped into the Profiler Component window.

Demo Version Limitations

Only modules are displayed and the Save function is disabled.

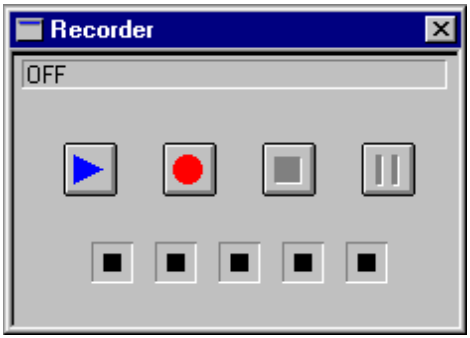
Associated Commands:

GRAPHICS on page 693, TUPDATE on page 734, DETAILS on page 673, RESET on page 716, BASE on page 656.

Recorder Component

The Recorder window shown in Figure 3.57 on page 100 provides record and replay facilities for debug sessions.

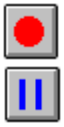
Figure 3.57 Recorder Window



The Recorder window enables the user to record and replay command files. The recorded file may also contain the time at which the command is executed.

Click the buttons shown below to record, play, pause and stop.

Play.

 **Record**.

 Stop.

 Pause.

An animation occurs during recording, replaying and pausing.

The current action (record, play or pause) and path of the involved file are displayed in the Object Infor bar of the window.

Recorder Operations

When there is no record or play session (e.g., when the window is open), only the record and play buttons are enabled.

When you click the record button, the debugger prompts you to enter a file name. Then a record session starts and the stop button is enabled. Click the stop button to end the record session.

Clicking the replay button prompts for a file name. Command files have a `.rec` default extension and can be edited. A replay session starts and only the stop and pause buttons are enabled. When the **pause** button is clicked, file execution stops and the play and stop buttons are enabled. When the play button is clicked, file execution continues from the point it has been stopped. When the **stop** button is clicked, the replay session stops.

Terminal and TestTerm Record

Data typed in the Terminal component and TestTerm component is recorded during a record session. The resulting file can be replayed only if the time is also recorded (**Record Time** menu entry of the recorder has to be checked before recording).

Recorder Menu

The Recorder menu shown in Figure 3.58 on page 101 changes according to the current session. The menu items are described in Table 3.30 on page 101.

Figure 3.58 Recorder Menu

Table 3.30 Recorder Menu Description

Menu Entry	Description
Record	Starts recording from a debug session.
Replay	Starts replaying from a debug session.
Record Time	If set, the evolution time is also recorded. Instant 0 corresponds to the beginning of the recording.

In Listing 3.2 on page 101, an .abs file is loaded, a breakpoint is set, the assembly component is configured to display the code and addresses. The Data1 component display is switched to local variables, and the application is started and stopped at the breakpoint.

Listing 3.2 Record File Example

```
at 4537 load C:\Freescale\DEMO\fibo.abs
at 9424 bs 0x1040 P
at 11917 Assembly < attributes code on
at 14481 Assembly < attributes adr on
at 20540 Data:1 < attributes scope local
at 24425 g
wait ;s
```

Drag Out:

Nothing can be dragged out.

Drop Into:

Nothing can be dropped in.

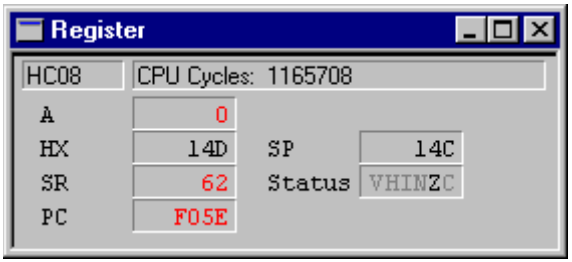
Demo Version Limitations

Only 20 commands are recorded and replayed.

Register Component

The Register window, shown in Figure 3.59 on page 103, displays the content of registers and status register bits of the target processor.

Figure 3.59 Register Window



Register values can be displayed in binary or hexadecimal format. These values are editable.

Status Register Bits

Set bits are displayed dark, whereas reset bits are displayed grey. Double-click a bit to toggle it. During program execution, contents of registers that have changed since the last refresh are displayed in red, except for status register bits.

The Object Infor bar of the window contains the number of CPU cycles as well as the processor's name.

Editing Registers

Double-click on a register to open an edit box over the register, so that the value can be modified.

Press the **ESC** key to ignore changes and retain previous content of the register.

If the **Enter** key is pressed outside the edited register, the new value is validated and the register content is changed.

If the **Tab** key is pressed, the new value is validated and the register content is changed. The next register value is selected and may be modified.

Double-clicking a status register bit toggles it.

Holding down the left mouse button and pressing the **A** key: Contents of Source, Assembly and Memory component windows change. The Source window shows the source code located at the address stored in the register. The Assembly window shows the disassembled code starting at the address stored in the register. The Memory window dumps memory starting at the address stored in the register.

Register Menu (Format Submenu)

The Register menu contains the items shown in Figure 3.60 on page 104. Table 3.31 on page 104 describes the menu entries.

Figure 3.60 Register Menu

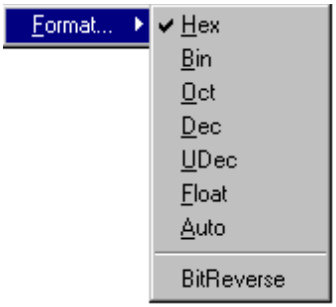


Table 3.31 Register Menu Description

Menu Entry	Description
Hex	Selects the hexadecimal register display format
Bin	Selects the binary register display format
Oct	Selects the octal register display format
Dec	Selects the signed decimal register display format
UDec	Selects the unsigned decimal register display format
Float	Selects the float register display format (all 32/64 bit registers are displayed as floats, all others as hex)
Auto	Selects the auto register display format (all floating point 32/64 bit registers are displayed as floats, all others as hex)
Bit Reverse	Selects the bit reverse data display format (Each bit is reversed).

Drag Out:

Table 3.32 on page 105 contains the drag actions possible from the Register window.

Table 3.32 Register Component Drag Possibilities

Destination Component Window	Action
Assembly	Assembly component receives an address range, scrolls up to the corresponding instruction and highlights it.
Memory	Dumps memory starting at the address stored in the selected register. The corresponding address is selected in the memory component.
Command Line	The address stored in the pointed to register is appended to the current command.

Drop Into:

Table 3.33 on page 105 shows the drop actions possible into the Register component.

Table 3.33 Register Component Drop Possibilities

Source Component Window	Action
Assembler	Loads the destination register with the PC of the selected instruction.
Data	Dragging the name loads the destination register with the start address of the selected variable. Dragging the value loads the destination register with the value of the variable.
Source	Loads the destination register with the PC of the first instruction selected.
Memory	Loads the destination register with the start address of the selected memory block.

Demo Version Limitations

No limitation

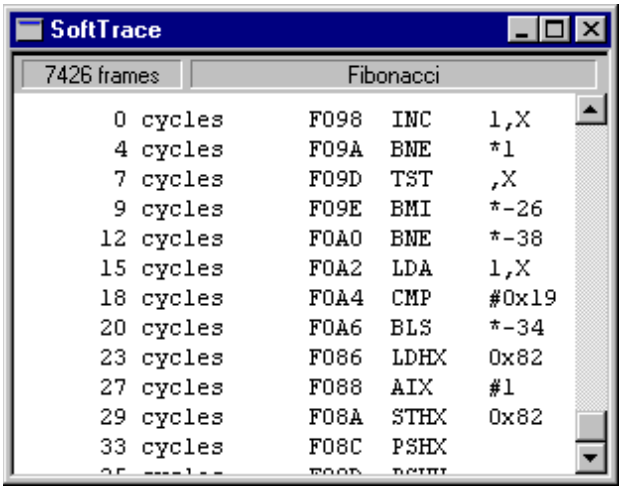
Associated Commands

ATTRIBUTES on page 645.

SoftTrace Component

The SoftTrace window shown in Figure 3.61 on page 106 records and displays instruction frames and time or cycles.

Figure 3.61 SoftTrace Window


SoftTrace
7426 frames
Fibonacci
0 cycles F098 INC 1,X
4 cycles F09A BNE *1
7 cycles F09D TST ,X
9 cycles F09E BMI *-26
12 cycles F0A0 BNE *-38
15 cycles F0A2 LDA 1,X
18 cycles F0A4 CMP #0x19
20 cycles F0A6 BLS *-34
23 cycles F086 LDHX 0x82
27 cycles F088 AIX #1
29 cycles F08A STHX 0x82
33 cycles F08C PSHX

The window's Object Bar displays the number of recorded frames and the name of the function where the selected frame is located.

SoftTrace Operations

Pointing at a frame and dragging the mouse forces all open windows to show the corresponding code or location. Time and cycles of all other frames are evaluated relative to this base.

Holding down the left mouse button and pressing the Z key sets the zero base frame to the pointed frame.

Holding down the left mouse button and pressing the D key forces all open component windows to show the code matching the pointed to frame.

SoftTrace Menu

The SoftTrace Menu shown in Figure 3.62 on page 107 contains the functions described in Table 3.34 on page 107.

Figure 3.62 SoftTrace Menu

Table 3.34 SoftTrace Menu Description

Menu Entry	Description
Record	Switches recording on and off.
Clock Speed	Sets the clock frequency.
Max Frames	Sets the maximum number of recorded frames. Therefore you can minimize the amount of memory required to display frames.
Cycles	Displays cycles instead of time (in ms).
ms	Displays time (in ms) instead of cycles.
Reset	Removes all recorded frames.

Associated Popup Menu

The SoftTrace popup menu shown in Figure 3.63 on page 107 contains functions (described in Table 3.35 on page 108) associated with the pointed to frame.

Figure 3.63 SoftTrace Associated Popup Menu

Table 3.35 SoftTrace Associated Popup Menu Description

Menu Entry	Description
Set Zero Base	Sets the zero base frame to the pointed to frame.
Show Location	Forces open component windows to show the code corresponding to the pointed to frame.

Drag Out:

Nothing can be dragged out.

Drop Into:

Nothing can be dropped in.

Demo Version Limitations

The number of frames is limited to 50.

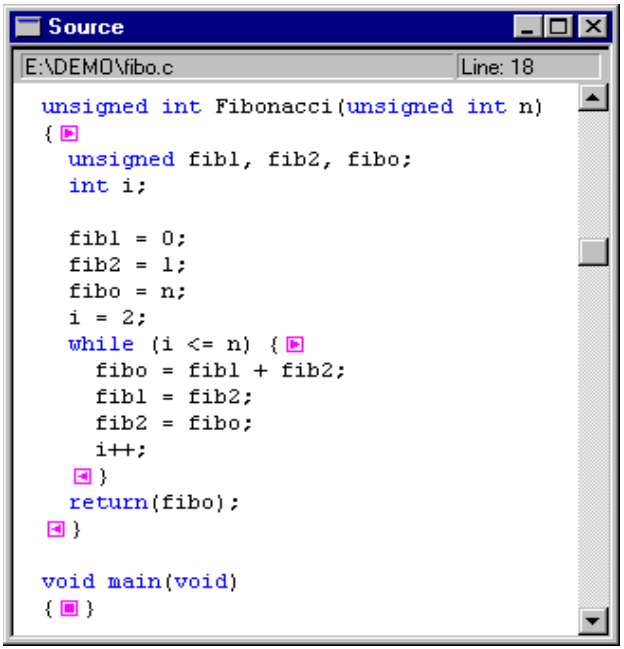
Associated Commands

CLOCK on page 665, CYCLE on page 667, FRAMES on page 688, RECORD on page 715, RESET on page 716.

Source Component

The Source window shown in Figure 3.64 on page 109 displays the source code of your program, i.e. your application file.

Figure 3.64 Source Window



The Source window allows you to view, change, monitor and control the current execution location in the program. The text displayed in the Source Component window is chroma-coded, i.e. language keywords, comments and strings are emphasized with different colors (respectively blue, green, red). A word can be selected by double-clicking it. A section of code can be selected by holding down the left mouse button and dragging the mouse.

The object info bar displays the line number in the source file of the first visible line that is at the top of the source.

Source code can be folded and unfolded. Marks (places where breakpoints may be set) can be displayed.

When the source statement matching the current PC is selected in this window, (e.g., in a C source: `fib1 = fib2`), the matching assembler instruction in the Assembler component window is also selected. This instruction is the next instruction to be executed by the CPU.

If breakpoints have been set in the program, they will be marked in the program source with a special symbol depending on the kind of breakpoint. For information on breakpoints refer to sections in the "Control Points" chapter. If execution has stopped, the current position is marked in the source component by highlighting the corresponding statement.

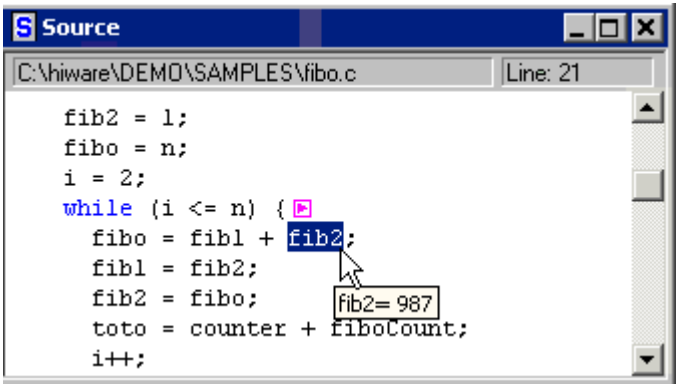
The complete path of the displayed source file is written in the Object Info bar of this window.

NOTE You cannot edit the visible text in the Source window. This is a file viewer only.

Tool Tips Features

The Debugger source component provides tool tips to display variable values. The tool tip is a small rectangular pop-up window that displays the value of the selected variable (shown in Figure 3.65 on page 110) or the parameter value and address of the selected procedure. A parameter or procedure can be selected by double-clicking it.

Figure 3.65 ToolTips Features



Select **ToolTips>Enable** from the source menu entry to enable or disable the tool tips feature.

Select **ToolTips>Mode** from the source menu entry to select normal or details mode, which provides more information on a selected procedure.

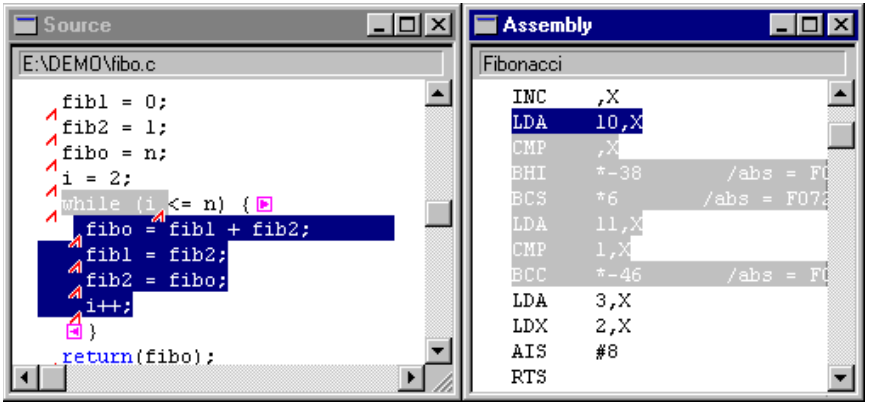
Select **ToolTips>Format** from the source menu entry to select the tool tip display format (Decimal, Hexadecimal, Octal, Binary or ASCII).

On Line Disassembling

For information about performing on line disassembly, refer to section How to Consult Assembler Instructions Generated by a Source Statement on page 203.

- Select a range of instructions in the source component and drag it into the assembly component. The corresponding range of code is highlighted in the Assembly component window, as shown in Figure 3.66 on page 111.
- Holding down the left mouse button and pressing the T key: Highlights a code range in the Assembly component window corresponding to the first line of code selected in the Source component window where the operation is performed. This line or code range is also highlighted.

Figure 3.66 On Line Disassembling

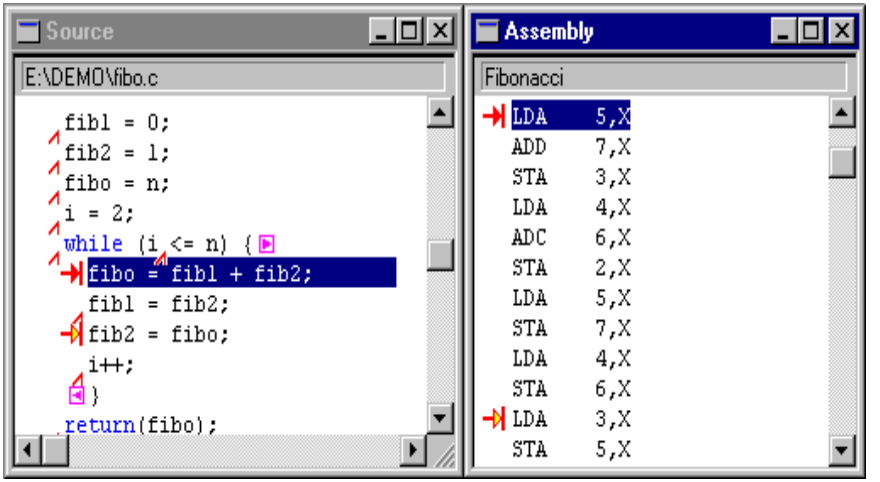


Setting Temporary Breakpoints

For information on how to set breakpoints refer to sections in the "Control Points" chapter.

- Point to an instruction in the Source component Window and click the right mouse button. The Source window popup menu is displayed. Select **Run To Cursor** from the popup menu. The application continues execution and stops at this location.
- Holding down the left mouse butoon and pressing the T key: Sets a temporary breakpoint at the nearest code position (visible with marks) thereafter the program runs and breaks at this location, as shown in Figure 3.67 on page 112.

Figure 3.67 Setting Breakpoints



Setting Permanent Breakpoints

- Point to an instruction in the Source component Window and click the right mouse button. The Source Component popup menu is displayed. Select **Set Breakpoint** from the popup menu. The permanent breakpoint icon is displayed in front of the source statement pointed to.
- Holding down the left mouse button and pressing the P key: Sets a permanent breakpoint at the nearest code position (visible with marks). The permanent breakpoint icon is displayed in front of the source statement pointed to.

Folding and Unfolding

Use this feature to show or hide a section of source code (e.g., source code of a function). For example, if a section is free of bugs, you can hide it. All text is unfolded at loading.

Sections of code that can be folded are enclosed between and .

Sections of code that can be unfolded are hidden under .

Double-click a folding mark or to fold the text located between the marks.

Double-click an unfolding mark to unfold the text that is hidden behind the mark.

Source Menus

The Source Menu ia shown in Figure 3.68 on page 113 and Figure 3.69 on page 114 shows the functions associated with the Source Popup Menu, while Table 3.36 on page 114 describes these functions.

Figure 3.68 Source Menu

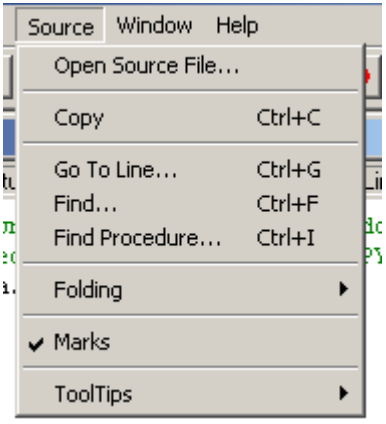


Figure 3.69 Source Associated Popup Menu

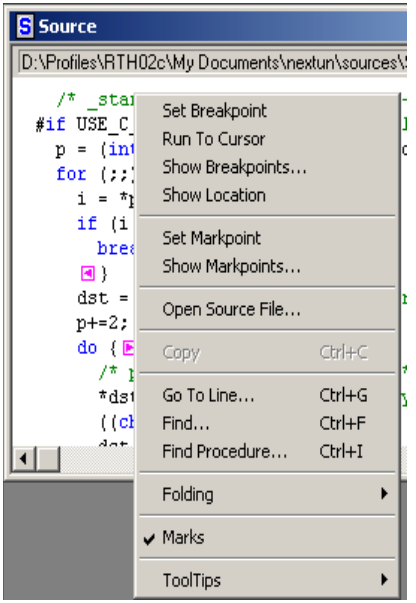


Table 3.36 Associated Pop - Up Menu Description

Menu Entry	Description
Set Breakpoint	Appears only in the Popup Menu if no breakpoint is set or disabled at the nearest code position (visible with marks). When selected, sets a permanent breakpoint at this position. If program execution reaches this statement, the program is halted and the current program state is displayed in all window components.
Delete Breakpoint	Appears only in the Popup Menu if a breakpoint is set or disabled at the nearest code position (visible with marks). When selected, deletes this breakpoint.
Enable Breakpoint	Appears only in the Popup Menu if a breakpoint is disabled at the nearest code position (visible with marks). When selected, enables this breakpoint.
Disable Breakpoint	Appears only in the Popup Menu if a breakpoint is set at the nearest code position (visible with marks). When selected, disables this breakpoint.

Table 3.36 Associated Pop - Up Menu Description (*continued*)

Menu Entry	Description
Run To Cursor	When selected, sets a temporary breakpoint at the nearest code position and continues program execution immediately. If there is a disabled breakpoint at this position, the temporary breakpoint will also be disabled and the program will not halt. Temporary breakpoints are automatically removed when they are reached.
Show Breakpoints	Opens the Controlpoints Configuration Window's Breakpoints Tab and allows you to view the list of breakpoints defined in the application and modify their properties (See "Control Points" chapter).
Show Location	Highlights a code range in the Assembly component window matching the line or selected source code. The line or the source code range are highlighted as well.
Set Markpoint	Appears only in the Popup Menu if a markpoint is disabled at the nearest code position (visible with marks). When selected, enables this markpoint.
Delete Markpoint	Appears only in the Popup Menu if a markpoint is set at the nearest code position (visible with marks). When selected, disables this markpoint.
Show Markpoints	Opens the Controlpoints Configuration Window's Markpoints Tab and allows you to view the list of markpoints defined in the application and modify their properties (See "Control Points" chapter).
Set Program Counter	The Program Counter is set to the address of the selected source code.
Open Source File	Opens the Source File Dialog if a CPU is loaded (see chapter below).
Copy (CTRL+C)	Copies the selected area of the source component into the clipboard. You can select a word by double-clicking it. You can select a text area with the mouse by moving the pointer to the left of the lines until it changes to a right-pointing arrow, and then drag up or down; automatic scrolling is activated when the text is not visible in the windows.
Go to Line (CTRL+G)	Opens a dialog box to scroll the window to a number line (see chapter below).

Table 3.36 Associated Pop - Up Menu Description (*continued*)

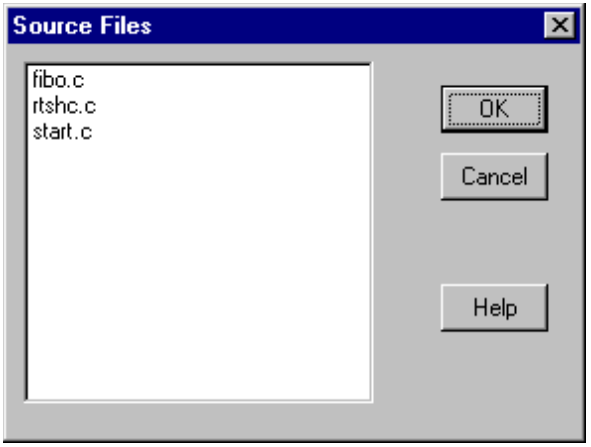
Menu Entry	Description
Find... (CTRL+F)	Opens a dialog box prompting for a string and then searches the file displayed in the source component. To start searching, click **Find Next**, the search is started at the current selection or at the first line visible in the source component (see chapter below).
Find Procedure (CTRL+I)	Opens a dialog box for searching a procedure (see chapter below).
Foldings	Opens the folding window (see chapter below)
Marks	Toggles the display of source positions where breakpoints may be set. If this switch is on, these positions are marked by small triangles.
ToolTips	Allows you to enable or disable the source tool tips feature, to set up the tool tip mode, and tool tip format.

NOTE If some statements do not show marks although the mark display is switched on, the following reasons may be the cause:
- The statement did not produce any code due to optimizations done by the compiler.
- The entire procedure was not linked in the application, because it is never used.

Open Source File

The Open Source File dialog box shown in Figure 3.70 on page 117 allows you to open the Source File (if a CPU is loaded). A source file is a file that has been used to build the currently loaded absolute file. Assembly file (*.dbg) is searched in the directory given by the OBJPATH and GENPATH variables. C, C++ files (`*.c,*.cpp,*.h,...`) are searched in the directories given by the GENPATH variable.

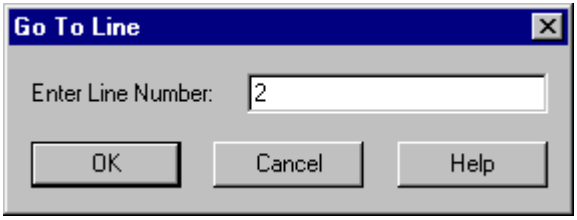
Figure 3.70 Open Source File Dialog Box



Go to Line

This menu entry is only enabled if a source file is loaded. It opens the dialog box shown in Figure 3.71 on page 118. In this dialog box, enter the line number you want to go to in the source component, the selected line will be displayed at the top of the source window. If the number is not correct, a message is displayed.

Figure 3.71 Go to Line Dialog Box



When this dialog box is open, the line number of the first visible line in the source is displayed and selected in the **Enter Line Number** edit box.

Find Operations

The Find dialog box, shown in Figure 3.72 on page 118 is used to perform find operations for text in the Source component. Enter the string you want to search for in the **Find what** edit box. To start searching, click **Find Next**, the search starts at the current selection or first line visible in the source component, when nothing is selected.

Use the **Up / Down** buttons to search backward or forward. If the string is found, the source component selection is positioned at the string. If the string is not found, a message is displayed.

Figure 3.72 Find Dialog Box

This dialog box allows you to specify the following options:

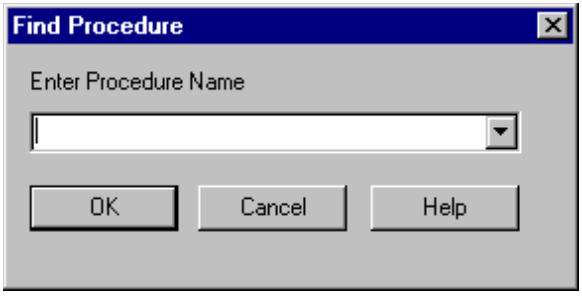
- **Match whole word only:** If this box is checked, only strings separated by special characters will be recognized.
- **Match case:** If this box is checked, the search is case sensitive.

NOTE If an item (single word or source section) has been selected in the Source component window before opening the Find dialog, the first line of the selection will be copied into the “Find what” edit box.

Find Procedure

The Find Procedure dialog box, shown in Figure 3.73 on page 119 is used to find the procedure name in the currently loaded application. Enter the procedure name you want to search for in the **Find Procedure** edit box. To start searching, click **OK**, the search starts at the current selection or at the first line visible in the source component, when nothing is selected.

Figure 3.73 Find Procedure Dialog Box



If a valid procedure name is given as a parameter, the source file where the procedure is defined is opened in the Source Component. The procedure's definition is displayed and the procedure's title is highlighted.

The drop-down list allows you to access the last searched items (classified from first to older input). Recent search items are stored in the current project file.

Folding Menu

The Folding Menu shown in Figure 3.74 on page 119 allows you to select the Fold functions described in Table 3.37 on page 120.

Figure 3.74 Folding Menu

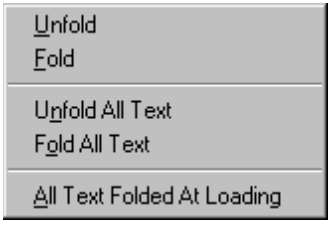


Table 3.37 Folding Menu Description

Menu Entry	Description
Unfold	Unfolds the displayed source code
Fold	Folds the displayed source code
Unfold All Text	Unfolds all displayed source code
Fold All Text	Folds all displayed source code
All Text Folded At Loading	Folds all source code at load time

Drag Out:

Table 3.38 on page 120 shows the drag actions possible from the Source component.

Table 3.38 Source Drag Possibilities

Destination Component Window	Action
Assembly	Displays disassembled instructions starting at the first high level language instruction selected. The assembler instructions corresponding to the selected high level language instructions are highlighted in the Assembly component
Register	Loads the destination register with the PC of the first instruction selected.
Data	A selection in the Source window is considered as an expression in the Data window, as if it was entered through the Expression Editor of the Data component. (please see Data Component on page 64 or Expression Editor on page 65)

Drop Into:

Table 3.39 on page 121 shows the drop actions possible into the Source component.

Table 3.39 Source Drop Possibilities

Source Component Window	Action
Assembly	Source component scrolls to the source statements corresponding with the pointed to assembly instruction and highlights it.
Memory	Displays high level language source code starting at the first address selected. Instructions corresponding to the selected memory area are greyed in the source component.
Module	Displays source code from the selected module.

Demo Version Limitations

Only one source file of the currently loaded application can be displayed.

Associated Commands

ATTRIBUTES on page 645, FIND on page 683, FOLD on page 686, FINDPROC on page 684, SPROC on page 727, SMOD on page 725, SPC on page 726, SMEM on page 724, UNFOLD on page 737.

Visualization Utilities

Besides components that provide the Debugger engine a well-defined service dedicated to the task of application development, the debugger component family includes utility components that extend to the productive phase of applications, such as, the host application builder components, process visualization components, etc.

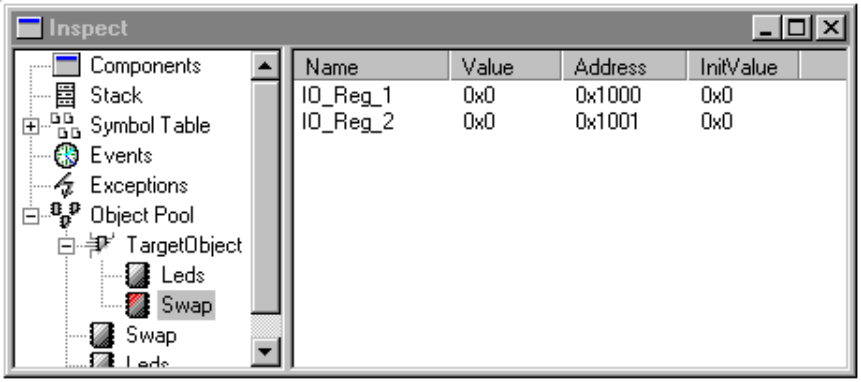
Among these components, there are visualization utilities that graphically display values, registers, memory cells, etc., or provide an advanced graphical user interface to simulated I/O devices, program variables, and so forth.

The following components of the continuously growing set of visualization utilities belong to the standard Debugger installation.

Inspector Component

The Inspector window shown in Figure 3.75 on page 122 displays information about several topics. It displays loaded components, the visible stack, pending events, pending exceptions and loaded I/O devices.

Figure 3.75 Inspector Component Window



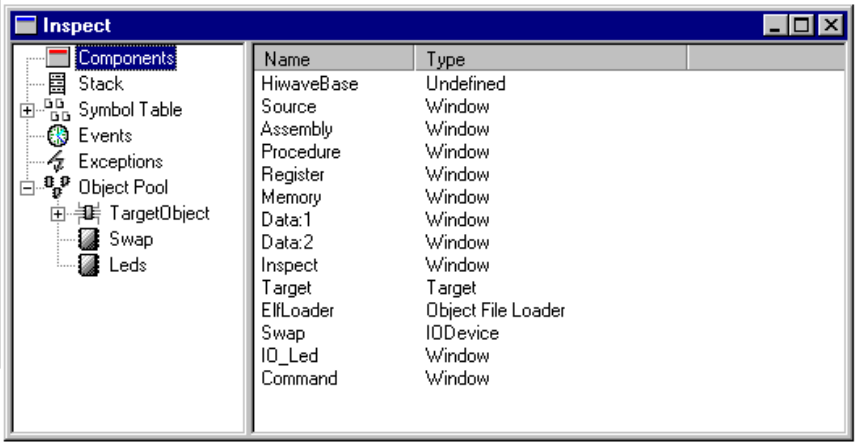
The hierarchical content of the items is displayed in a tree structure. If any item is selected on the left side, then additional information is displayed on the right side.

In the figure above, for example, the Object Pool is expanded. The Object Pool contains the TargetObject, which contains the Leds and Swap peripheral devices. The Swap peripheral device is selected and registers of the Swap device are displayed.

Components Icon

When the components icon is selected in the Inspect window, as shown in Figure 3.76 on page 123, the right side displays various information about all loaded components. A Component is the “unit of dynamic loading”, therefore all windows, the CPU, the connection and maybe the connection-simulator are listed.

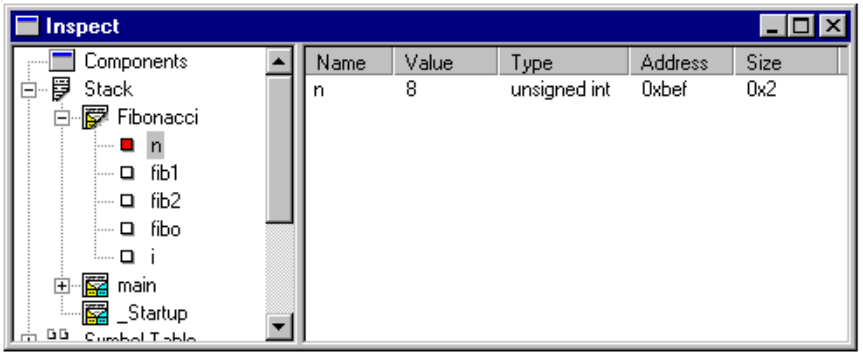
Figure 3.76 Inspect Window Components Icon



Stack Icon

The Stack icon shown in Figure 3.77 on page 123 displays the current stack trace. Every function on the stack has a separate icon on the trace. In the stack-trace, the content of a local variable is accessible.

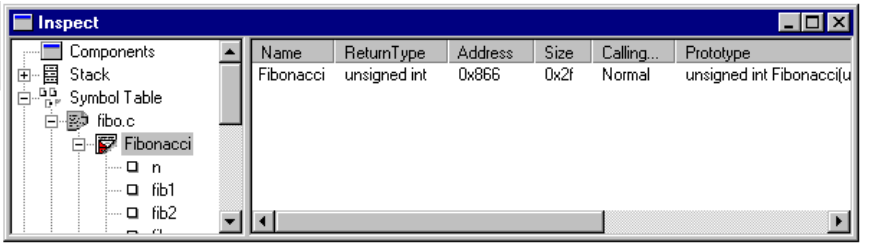
Figure 3.77 Inspector Window Stack Icon



Symbol Table

The symbol table shown in Figure 3.78 on page 124 displays all loaded symbol table information in raw format. There are no stack frames associated with functions. Therefore the content of local variables is not displayed. Global variables and their types are displayed.

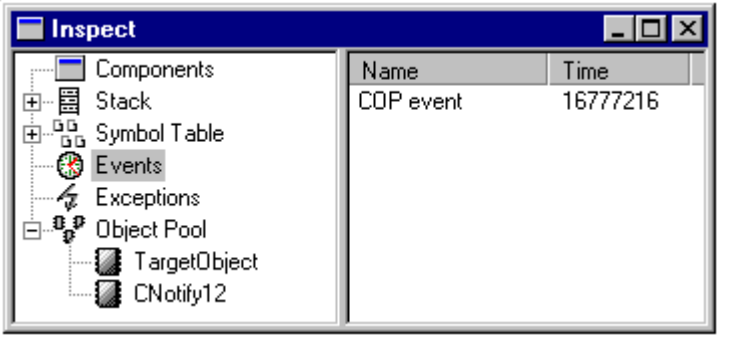
Figure 3.78 Inspector Window Symbol Table



Events Icon

The Inspect window Events icon shown in Figure 3.79 on page 124 shows all currently installed events. Events are handled by peripheral devices, and notified at a given time. The Event display shows the name of the event and remaining time until the event occurs.

Figure 3.79 Inspector Window Events Icon



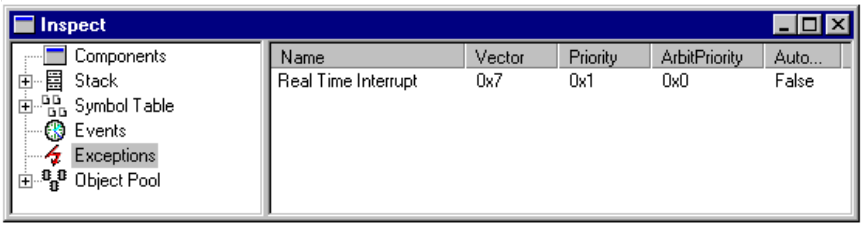
Events are only used in the HC(S)12(X) Freescale Full Chip Simulator. This information is used for simulation I/O device development.

When simulating a watchdog/COP, an event with the remaining time is displayed in the Event View.

Exceptions Icon

The Inspector window Exceptions icon shown in Figure 3.80 on page 125 shows all currently raised exceptions. Exceptions are pending interrupts.

Figure 3.80 Inspector Window Exceptions Icon



Events are only used in the HC(S)12(X) Freescale Full Chip Simulator. This information is used for simulation I/O device development.

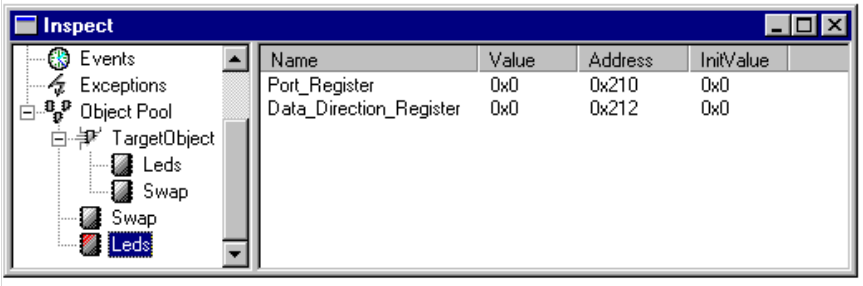
Since interrupts are usually simulated immediately when they are raised, the Exceptions are usually empty. Only when interrupts are disabled or an interrupt is handled, something is visible in this item.

When simulating a watchdog/COP, an Exception is raised as soon as the watchdog time elapses.

Object Pool

The Object Pool shown in Figure 3.81 on page 125 is a pool of objects. It can contain any number of Objects, which can communicate together and also with other parts of the Debugger.

Figure 3.81 Inspector Window Object Pool



The most common use of Objects is to simulate special hardware with the I/O development package, however, other connections also use the Object Pool. For example, the Terminal Component exchanges its input and output by the Object Pool. The Terminal Component also operates with some hardware connections.

For the HC(S)12(X) Freescale Full Chip Simulator, the Object Pool usually contains the TargetObject, which represents the address space. All Objects that are loaded are displayed in the Object Pool. The TargetObject additionally shows the objects that are mapped to the address space.

Inspector Operations

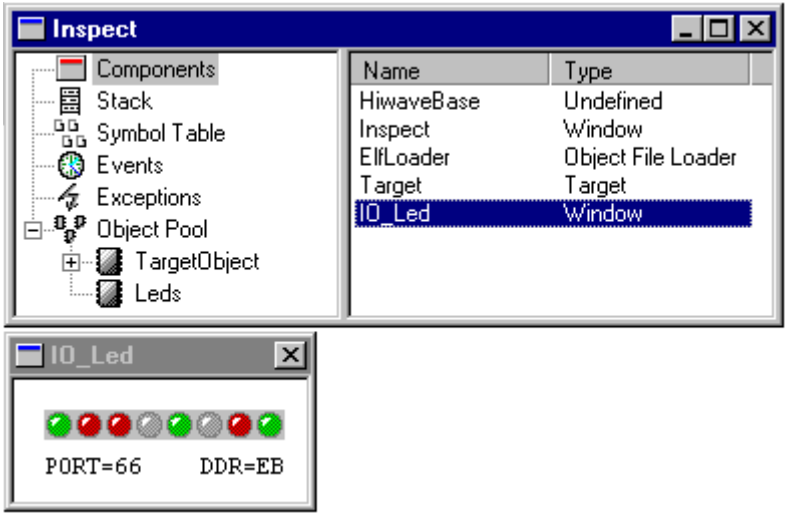
Click the folded/unfolded icons to unfold/fold the tree and display/hide additional information.

Click on any icon or name to see the corresponding information displayed on the right side.

On the right side, some value fields can be edited by double clicking on them. Only values that are accessible can be edited. Usually, if a value is displayed, it can be changed. I/O Devices in the Object Pool do not accept all new values, depending on the I/O Device. Values can be entered in hexadecimal (with preceding **0x**), in decimal, in octal (with preceding **0**), or in binary (with preceding **&**).

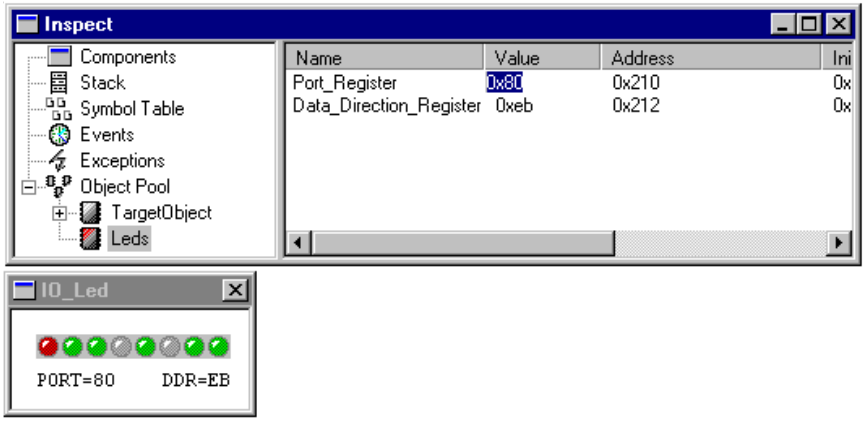
To see the IO_Led in the Inspector, as shown in Figure 3.82 on page 126, open the IO_Led with the context menu **Component**-Open and then open the Inspector. If the Inspector is already loaded, select **Update** from the context menu in the Inspector. Then click on the Components icon to see the Component list, which now includes the "IO_Led" component.

Figure 3.82 How to See the IO_Led in the Inspector Window



Expand Object Pool, to see the Leds icon. Click on the Leds icon. On the right side, the Port_Register and Data_Direction_Register are displayed with their current value. Double click on the values to change them (Figure 3.83 on page 127).

Figure 3.83 Changing “Data_Direction_Register” Value



Inspector Menu

The Inspector menu contains entries described in Table 3.40 on page 127.

Table 3.40 Inspector Menu Entries

Menu Entry	Description
Update	All displayed information is updated Items that no longer exist are removed and new items are added.

Associated Popup Menu

Commands in the Inspector context menu depend on the selected item. It can contain entries described in Table 3.41 on page 128.

Table 3.41 Inspector Popup Menu Entries Description

Menu Entry	Context	Description
Update	All items	All displayed information is updated Items that no longer exist are removed and new items are added.
Max. Elements...	All items	To display large arrays element by element, the maximum number can be configured. It is also possible to display a dialog that prompts the user.
Format	All items	Numerical values can be displayed in different formats.
Close	single selected Component only	Closes the corresponding component

Drag Out:

Items that can be dragged, depends on which icon is selected. Table 3.42 on page 128 gives a brief description.

Table 3.42 Inspector Component Drag Possibilities

Dragging Item	Description
Components	The components cannot be dragged
Stack	The Stack Icon itself cannot be dragged. All subitems can be dragged the same way as the Symbol Table subitems, described below.
Symbol Table	The Symbol Table icon cannot be dragged out. Subitems can be dragged depending on their type: Modules: Modules can be dragged to the source and global data window to specify a specific module. Functions: Functions can be dragged to display the function or code range. Variables: Variables can be dragged to display their content in memory. Indirections: Indirections can be dragged to display their content in memory.

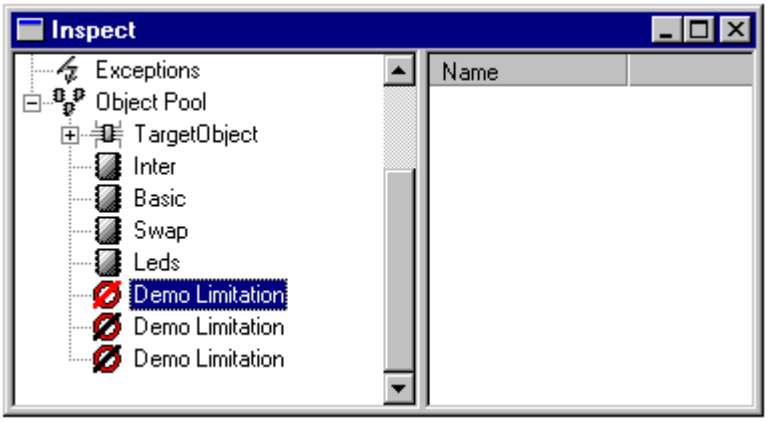
Drop Into:

Nothing can be dropped in.

Demo Version Limitations

Only 5 items can be expanded at each location. For remaining items, an icon with the text “Demo Limitation” is displayed, as shown in Figure 3.84 on page 129.

Figure 3.84 Inspector Component Demo Version Limitations

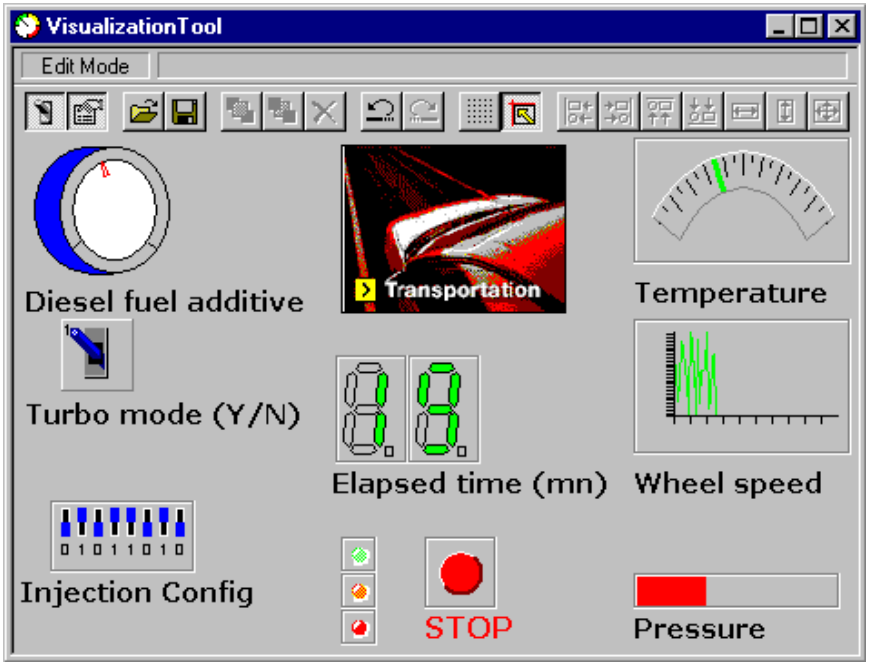


VisualizationTool Component

The VisualizationTool component is a very convenient tool for presenting your data. For software demonstration, or for your own debugging session, take advantage of all its virtual instruments.

The VisualizationTool window, shown in Figure 3.85 on page 130, consists of a plain workspace that can be equipped with many different instruments.

Figure 3.85 VisualizationTool Window



Edit Mode and Display Mode

The VisualizationTool may operate in two modes: Display mode or Edit mode.

The Edit mode is for designing the workspace to suit your needs. In the Display mode you can then use what you have done in the Edit mode, that is, to view values, interact with your application and instruments, press buttons, etc.

To switch between these two modes, you can use the toolbar, the context menu, or the shortcut Ctrl+E.

Add New Instrument

Use the context menu (VisualizationTool Menu on page 131 on page 131) to add a new instrument.

Instrument Selection

You can select a single instrument by left clicking the mouse on it, and change the selection by pressing the tab-key.

To make multiple selections, hold down the control key and left-click on the desired instruments. You can also left click, hold and move to create a selection rectangle.

Move Instruments

There are two ways to move instruments. First, make your desired selection. You can then use the mouse to drag the instruments, or use the cursor keys to move them step by step (hold down the control key to move the instrument in steps of ten). The move process performed with the mouse can be broken off by pressing the escape key.

Resize Instruments

When you select a instrument, sizing handles appear at the corners and along the edges of the selection rectangle. You can resize an object by dragging its sizing handles, or by using the cursor keys while holding down the shift key. The resize process performed with the mouse can be broken off by pressing the escape key. Only one instrument can be resized at a time. Furthermore, each instruments has its own size minimum.

VisualizationTool Menu

Once the Visualization Tool component has been launched, its menu appears in the debugger menu bar. The menu contains the entries described in Table 3.43 on page 131.

Table 3.43 Visualization Tool Menu Description

Menu Entry	Description
Properties	Displays the properties of the currently selected instrument. Shortcut: <Ctrl+P>
Add New Instrument	Enables to choose an instrument from the list and add it to the view.
Paste	Pastes an instrument that has been previously copied. Shortcut: <Ctrl+V>
Select All	Selects all the instruments of the view. Shortcut: <Ctrl+A>
Edit mode	Switches between Display mode and Edit mode. In Edit mode, this entry is checked. Shortcut: <Ctrl+E>

Table 3.43 Visualization Tool Menu Description (*continued*)

Menu Entry	Description
Load Layout	Loads a VisualizationTool-Layout (`*.vtl`). The actual instruments will not be removed. Shortcut: <Ctrl+L>
Save Layout	Saves the current layout to a file (`*.vtl`). Shortcut: <Ctrl+S>

Associated Popup Menu

The context menu of the VisualizationTool depends on the current selection. It can contains the entries described in Table 3.44 on page 132.

Table 3.44 VisualizationTool Popup Menu

Menu entry	Context	Description
Edit mode	Always	Switches between Display mode and Edit mode. In Edit mode, this entry is checked.
Setup	Always	Shows the Setup dialog of the VisualizationTool.
Load Layout	Edit mode	Loads a VisualizationTool-Layout (`*.vtl`).
Save Layout	Always	Saves the current layout to a file (`*.vtl`).
Add New Instrument	Edit mode	Shows a new popup menu with all available instruments.
Properties	Only one instrument selected	Shows up the property dialog box for the currently selected instrument. Shortcut: Ctrl + P
Remove	At least one selection	Removes all currently selected instruments. Shortcut: Delete
Copy	At least one selection	Copies the data of the currently selected instruments into the clipboard. Shortcut: Ctrl + C

Table 3.44 VisualizationTool Popup Menu (*continued*)

Menu entry	Context	Description
Cut	At least one selection	Cuts the currently selected instruments into the clipboard. Shortcut: Ctrl + X
Paste	Edit mode	Adds instruments, which are temporary stored in the clipboard, to the workspace. Shortcut: Ctrl + V
Send to Back	At least one selection	Sends the current instrument to the back of the Z-order.
Send to Front	At least one selection	Brings the current instrument to the front of the Z-order.
Clone Attributes	More than one selection	Clones the common attributes to all selected instruments according to the last selected. Shortcut: <Ctrl + Enter>
Align	At least two selections	Gives access to a new menu for alignment.
Top	Align	Aligns the instruments to the top line of the last selected instrument.
Bottom	Align	Aligns the instruments to the bottom line of the last selected instrument.
Left	Align	Aligns the instruments to the left line of the last selected instrument.
Right	Align	Aligns the instruments to the right line of the last selected instrument.
Size	Align	Makes the size of all selected instruments the same as the last selected.
Vertical Size	Align	Makes the vertical size of all selected instruments the same as the last selected.
Horizontal Size	Align	Makes the horizontal size of all selected instruments the same as the last selected.

VisualizationTool Properties

Like other instruments, the VisualizationTool itself has got Properties. There are several configuration possibilities for the VisualizationTool, shown in Table 3.45 on page 134. To view the property dialog box of the VisualizationTool, use the shortcut <CTRL-P> or double click on the background.

Table 3.45 VisualizationTool Properties

Menu Entry	Description
Edit mode	Switches from Edit mode to Display mode.
Display Scrollbars	Switches the scrollbars on, off, or sets it to automatic mode.
Display Headline	Switches the headline on or off.
Backgroundcolor	Specifies the background color of the VisualizationTool.
Grid Mode	Specifies the grid mode. There are four possibilities: 'Off,' 'Show grid but no snap,' 'Snap to grid without showing the grid,' or 'Show the grid and snap on it.'
Grid Size	Specifies the distance between two grid points (vertical, horizontal).
Grid Color	Specifies the color of the grid points.
Refresh Mode	Specifies the way the window will be refreshed. You may choose between: "Automatic, Periodical, Each access, Cpu Cycles".

Instruments

When you first add an instrument, it is in "move mode". Place it at the desired location on the workspace. All new instruments are set to their default attributes. To configure an instrument, right-click on an instrument and choose 'Properties', or double click on it. All instruments have the common attributes shown in Table 3.46 on page 134.

Table 3.46 Instruments Properties Attributes

Attribute	Description
X-Position	Specifies the X-coordinate of the upper left corner.
Y-Position	Specifies the Y-coordinate of the upper left corner.
Height	Specifies the instruments height.

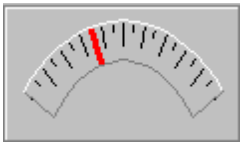
Table 3.46 Instruments Properties Attributes (*continued*)

Attribute	Description
Width	Specifies the instruments width.
Bounding Box	Specifies the look of the bounding box. Available displays are: No Box, Flat (outline only), Raised, Sunken, Etched, and Shadowed.
Backgroundcolor	Defines the color of the instrument's background. The checkbox enables to set a color or let the instrument be transparent.
Kind of Port	Specifies the kind of port to be used to get the value to display. The location must be specified in the 'Port to Display' field.
Port to Display	Defines the location of the value be used for the instrument's visualization. Here are some Examples: Substitute: *TargetObject.#210* Subscribe: *TargetObject.#210* Subscribe: *PORTB.PORTB (check exact spelling using Inspector)* Variable: *counter* Register: *SP* Memory: *0x210*
Size of Port	If you use the Memory Port, you can also specify the width of memory to display (up to 4 Bytes).

Analog Instrument

The Analog instrument (Figure 3.86 on page 135) represents the classical pointer instrument, also known as speedometer, voltage meter...

Figure 3.86 Analog Instrument



Analog instrument attributes are shown in the Table 3.47 on page 136.

Table 3.47 Analog Instrument Attributes

Attribute	Description
Low Display Value	Defines the zero point of the indicator. The values below this definition will not be displayed.
High Display Value	Defines the highest position of the indicator. It defines the value on which the indicator reads 100%.
Indicatorlength	Defines the length of the small indicator. The minimal value is set to 20.
Indicator	Defines the color of the indicator. The default color is red.
Marks	Defines the color of the marks. The default color is black.

Bar Instrument

Using the Bar instrument, values are displayed by a bar strip. This instrument (See Figure 3.87 on page 136) may be used as a position state of a water tank.

Figure 3.87 Bar Instrument

Bar instrument attributes are shown in the Table 3.48 on page 136

Table 3.48 Bar Instrument Attributes

Attribute	Description
Low Display Value	Defines the zero point of the indicator. The values below this definition will not be displayed.
High Display Value	Defines the highest position of the indicator. It defines the value on which the indicator reads 100%.
Bardirection	Sets the desired direction of the bar that displays the value.
Barcolor	Specifies the color of the bar. Default color is red.

Bitmap Instrument

You can use the Bitmap instrument to give a special look to your visualization, or to display a warning picture.

Figure 3.88 Bitmap Instrument

Additionally, it can also be used as a bitmap animation. Its attributes are shown in the Table 3.49 on page 137

Table 3.49 Bitmap Instrument Attributes

Attribute	Description
Filename	Specifies the location of the bitmap. With the button behind, you can browse for files.
AND Mask	Performs a bitwise-AND operation with this value. AND the value of the selected port. Default value is 0.
EQUAL Mask	This value is compared to the result of the AND operation. The bitmap is displayed only if both values are the same. Default value is 0.

In general, for showing the bitmap, following condition has to be true:
(port_memory & ANDmask) == EQUALmask

A practical example about using the AND and EQUAL masks is following example:
You want to show in the visualization a taillight of a car. for this you need bitmaps (e.g. from a digital camera) of all possible states of the taillight (e.g. flasher on, brake light on, etc.). Usually the status of all lamps are encoded into a port or memory cell in your application, and each bit in this cell describes if a lamp is on or not. E.g. bit 0 says that the flasher is on, where bit 1 says that the brake light is on. So for your simple application you need following bitmaps with their settings:
- no light on bitmap: AND mask 3, EQUAL mask 0
- flasher on bitmap: AND mask 3, EQUAL mask 1
- brake light on bitmap: AND mask 3, EQUAL mask 2
- brake and flasher light on: AND mask 3, EQUAL mask 3

DILSwitch Instrument

The DILSwitch instrument is also known as Dual-in-Line Switch (Figure 3.89 on page 138). It is mainly used for configuration purpose. You can use it for viewing or setting bits of one to four bytes.

Figure 3.89 DILSwitch Instrument

OILSwitch instrument attributes are listed in the Table 3.50 on page 138.

DIL Switch instruments attributes

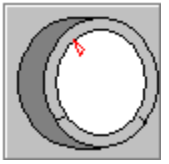
Table 3.50 DILSwitch Instrument Attributes

Attribute	Description
Display 0/1	When enabled, displays the value of the bit under each plot of the DILSwitch instrument.
Switch Color	Specifies the color of the switch.

Knob Instrument

The Knob instrument is normally known as an adjustment instrument. For example, in can simulate the volume control of a radio (Figure 3.90 on page 138).

Figure 3.90 Knob Instrument



Knob instrument attributes are shown in the Table 3.51 on page 138

Table 3.51 Knob Instrument Attributes

Attribute	Description
Low Display Value	Defines the zero point of the indicator. The values below this definition will not be displayed.
High Display Value	Defines the highest position of the indicator. It defines the value on which the indicator reads 100%.

Table 3.51 Knob Instrument Attributes (*continued*)

Attribute	Description
Indicator Color	Defines the color and the width of the pen used to draw the indicator.
Knob Color	Defines the color of the knob side.

LED Instrument

The LED instrument is used for observing one definite bit of one byte (Figure 3.91 on page 139). There are only two states: On and Off.

Figure 3.91 Led Instrument

LED instrument attributes are shown in Table 3.52 on page 139.

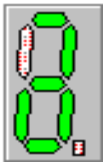
Table 3.52 LED Instrument Attributes

Attribute	Description
Bitnumber to Display	Defines the bit of the given byte to be displayed.
Color if Bit = = 1	Defines the color if the given bit is set.
Color if Bit = = 0	Defines the color if the given bit is not set.

7-Segment Display Instrument

This is the well known 7-Segment Display instrument for numbers and characters. It has seven segments and one point. These eight units represent eight bits of one byte (Figure 3.92 on page 139).

Figure 3.92 7-Segment Display Instrument



7 Segment Display instrument attributes are shown in Table 3.53 on page 140

Table 3.53 7 Segment Display Instrument Attributes

Attribute	Description
Decimalmode	Displays the first four or the second four bits of one byte in hexadecimal mode. When it is switched off, each segment will represent one bit of one byte.
Sloping	Switches the sloping on or off.
Display Version	Selects the appearance of the instrument. There are two versions available.
Color if Bit = = 1	Defines the color of an activated segment. You may also set the color to transparent.
Color if Bit = = 0	Defines the color of a deactivated segment. You may also set the color to transparent.
Outlinecolor	Defines the color of the segment outlines. You may also set the color to transparent.

Switch Instrument

Use the Switch instrument to set or view a definite bit (Figure 3.93 on page 140). The Switch instrument also provides an interesting debugging feature: you can let it simulate bounces, and thus check whether your algorithm is robust enough. Four different looks of the switch are available: slide switch, toggle switch, jumper or push button.

Figure 3.93 Switch Instrument

Switch instrument attributes are shown in Table 3.54 on page 140.

Table 3.54 Switch Instrument Attributes

Attribute	Description
Bitnumber to Display	Specifies the number of the bit you want to display.
Display 0/1	Enables to display the value of the bit in its upper left corner.
Top Position is	Specifies if the 'up' position is either zero or one. Especially useful to easily transform the push button into a reset button.

Table 3.54 Switch Instrument Attributes (*continued*)

Attribute	Description
Kind of Switch	Changes the look of the instrument. Following kinds of switches are available: Slide Switch, Toggle Switch, Jumper, Push Button. The behavior of the Push Button slightly differs from the others, since it returns to its initial state as soon as it has been released.
Switch Color	Specifies the color of the switch.
Bounces	If enabled, gives access to the following other attributes to configure the way the switch will bounce.
Nb Bounces	Specifies the number of bounces before stabilization.
Bounces on Edge	Specifies wether the switch will bounce on falling, rising or both edges.
Type of Unit	Synchronizes the frequency of the bouncing either on the timer of your host machine, or on CPU cycles.
Pulse Width (100ms)	Defines the duration of one bounce. This attribute should be filled in if you chose “Host Periodical” in the “Type of Unit” attribute.
CPU Count	This attribute represents the number of CPU cycles to reach before the switch changes its state. It should be filled in if you chose “CPU Cycles” in the “Type of Unit” attribute.

Text Instrument

The Text instrument has several functions: Static Text, Value, Relative Value, and Command (Figure 3.94 on page 141).

Figure 3.94 Text Instrument

Please use ’Text Mode’ to switch between the five available modes. Text instrument common attributes are shown in the Table 3.55 on page 142

Table 3.55 Text Instrument Attributes

Attribute	Description
Text Mode	Specifies the mode. Choose among four modes : Static Text, Value, Relative Value, and Command
Displayfont	Defines the desired font. All installed Windows fonts are available.
Horiz. Text Alignment	Specifies the desired horizontal alignment of the text in the given bounding box.
Vert. Text Alignment	Specifies the desired vertical alignment of the text in the given bounding box.
Textcolor	Defines the color of the given text.

'Static Text' is used for adding descriptions on the workspace. Its attributes are shown in Table 3.56 on page 142.

Table 3.56 Static Text Attributes

Attribute	Description
Field Description	Contains the text to be displayed.

'Value' is used for displaying a value in different ways (decimal, hexadecimal, octal, or binary). Its attributes are shown in Table 3.57 on page 142.

Table 3.57 Value Attributes

Attribute	Description
Field Description	Contains the additional description that will be displayed in front of the value. Add a colon and/or space as you wish. The default setting is "Value: "
Format mode	Defines the format. Choose among this list: Decimal, Hexadecimal, Octal, and Binary formats.

'Relative Value' is used for showing a value in a range of 0 up to 100% or 1000‰. Its attributes are shown in Table 3.58 on page 143

Table 3.58 Relative Value Attributes

Attribute	Description
Field Description	Add the additional description text to be displayed in front of the value. Add a colon and/or space if desired. The default setting is "Value: "
Low Display Value	Fixes the minimal value that will represent 0%. Values below this definition will appear as an error: #ERROR.
High Display Value	Fixes the maximal value that will represent 100%. Values above this definition will appear as an error: #ERROR..
Relative Mode	Switches between percent and permill.

'Command'. With this instrument mode you can specify a command that will be executed by clicking on this field. For more information about commands, read the chapter 'Debugger Commands'. Command mode attributes are shown in the Table 3.59 on page 143

Table 3.59 Command Attributes

Attribute	Description
Field Description	Contains the text that will be displayed on the button.
Command	Contains the command-line command to be executed after pressing the button.

'CMD Callback' This mode is the same as command, but with one difference: The returned value will be shown as text instead of 'Field Description'. Its attributes are shown in Table 3.60 on page 143

Table 3.60 CMD Callback Attributes

Attribute	Description
Field Description	Warning: there is no use to fill out his field as the text will be overwritten the first time you execute the specified command.
Command	Contains the command line command to be executed after pressing the button.

Drop Into:

In Edit mode, the drag and drop functionality supplies a very easy way to automatically configure an instrument.

To assign a variable, simply drag it from the Data Window onto the instrument.

The “kind of Port” is immediately set on “Memory” and the “Port to Display” field contains now the address of the variable. Now repeat the drag-and-drop on a bare portion of the VisualizationTool window: a new text instrument is created, with correct port configuration.

Some other components allow this operation:

- The memory window: select bytes and drag-and-drop them onto the instrument.
- The Inspector component: pick an object from the object pool.

Demo Version Limitations

If you work in demo mode, you will only be able to load one VisualizationTool window. The number of instruments is limited to three.

4

Control Points

This chapter provides an overview of the debugger control points: Breakpoints, Watchpoints, and Markpoints. Click any of the following links to jump to the corresponding section of this chapter:

- Introduction on page 145
- Breakpoints on page 146
- Setting Breakpoints on page 152
- Watchpoints on page 159
- Setting Watchpoints on page 163
- Markpoints on page 169
- Setting Markpoints on page 172
- Halting on a Control Point on page 174

Introduction

There are three kinds of control points:

Breakpoints (also called data breakpoints): Breakpoints are located at an address, They can be temporary or permanent.

Watchpoints: Watchpoints are located at a memory range. They start from an address, have a range, and a read and/or write state.

Markpoints: Are marked points of observation that can be jumped to by the programmer. They can be located in data, source or memory.

You can set or disable a control point, set a condition and an optional command, and set the current count and counting interval, using the popup menu of the Source, Memory or Assembly window.

You can see and edit control point characteristics through the three tabs of the Controlpoints Configuration Window: Breakpoint, Watchpoints and Markpoints tabs. These three tabs have common properties that allow you to interactively perform the following operations on control points:

- Selecting a single control point from a list box and clicking **Delete**.
- Selecting multiple control points from a list box and clicking **Delete**.

- Enabling/disabling a selected control point by checking or unchecking the related checkbox.
- Enabling/disabling multiple control points by checking or unchecking the related checkbox.
- Enter or modify the condition of a selected control point.
- Enabling/disabling the condition of a selected control point by checking/unchecking the related checkbox.
- Enter or modify the command of a selected control point.
- Enabling/disabling the command of a selected control point by checking/unchecking the related checkbox.
- Enabling/disabling multiple control point commands by selecting control points and checking/unchecking the related checkbox.
- Modifying the counter and/or limit of a single control point.

With breakpoints, the following operations are also available:

- Enabling/disabling halting on a single temporary breakpoint by checking/unchecking the matching checkbox.
- Enabling/disabling halting on multiple temporary breakpoints by checking/ unchecking the matching checkbox.

With watchpoints, the following operations are also available:

- Enabling/disabling halting on a single read and/or write access by checking/ unchecking the corresponding checkboxes.
- Enabling/disabling halting on multiple read and/or write accesses by checking/ unchecking the corresponding checkboxes.
- Defining the memory range controlled by the watchpoint.

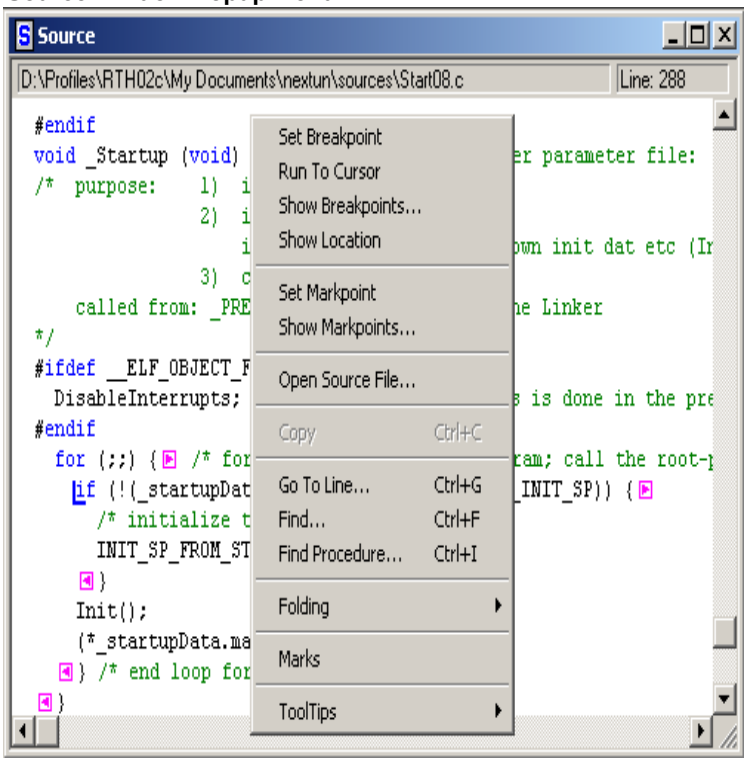
Breakpoints

Breakpoints are control points associated with a PC value. That is, program execution is stopped as soon as the PC reaches the value defined in a breakpoint. The Debugger supports four different types of breakpoints:

- Temporary breakpoints, which are activated next time the instruction is executed.
- Permanent breakpoints, which are activated each time the instruction is executed.
- Counting breakpoints, which are activated after the instruction has been executed a certain number of times.
- Conditional breakpoints, which are activated when a given condition is TRUE.

Breakpoints are controlled through the Breakpoints tab of the Controlpoints Configuration window. This window can be opened through the Source Window Popup menu, as described below:

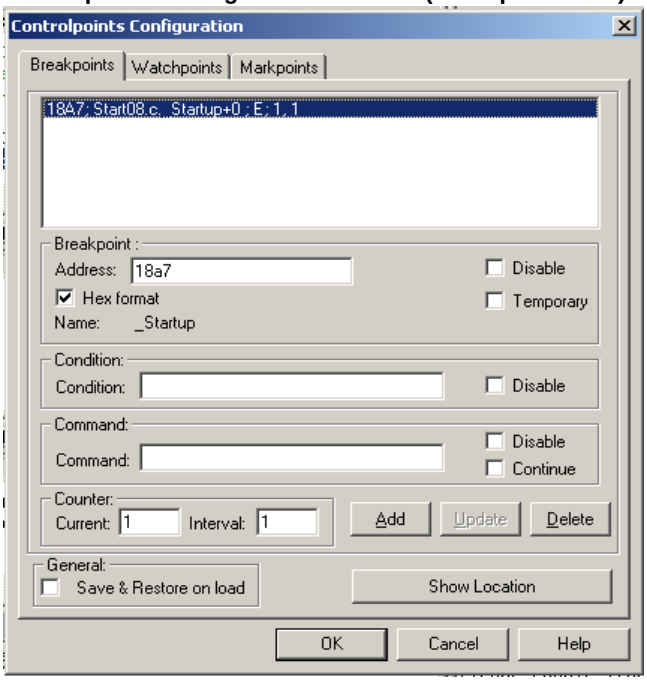
Figure 4.1 Source Window Popup Menu



1. Point at a C statement in the Source window, and click the right mouse button
2. Select **Show Breakpoints** from this menu

The Controlpoints Configuration Window (Breakpoints Tab) on page 148 is opened. The Breakpoints tab of this window is shown in Figure 4.2.

Figure 4.2 Controlpoints Configuration Window (Breakpoints Tab)



Breakpoints Tab

The Controlpoints Configuration Window (Breakpoints Tab) on page 148 contains:

- List box that displays the list of currently defined breakpoints
- "**Breakpoint**:" group box that displays the address of the currently selected breakpoint, name of procedure in which the breakpoint has been set, state of the breakpoint (disabled or not), and type of breakpoint (temporary or permanent).
- "**Condition**:" group box that displays the condition string to evaluate, and the state of the condition (disabled or not).
- "**Command**:" group box that displays the command string to execute and the state of the command (disable or continue after command execution).
- "**Counter**:" group box that displays the current value of the counter and interval value of the counter.

NOTE Current and Interval values are limited to 2,147,483,647; if entering a number greater than this value, a beep occurs and the character is not appended. When the Interval value is changed, the Counter value is automatically set to the Interval value.

- "**Delete**" button to remove the currently selected breakpoint.
- "**Update"** button to Update all modifications in the dialog.
- "**Add"** button to add new breakpoints; specify the Address (in hexadecimal when **Hex format** is checked, or as an expression when **Hex format** is unchecked).
- "**OK"** button to validate all modifications.
- "**Cancel"** button to ignore all modifications.
- "**Help"** button to open related help information.

Multiple Selections in List Box

The list box allows you to select multiple consecutive breakpoints by clicking the first breakpoint then pressing the **Shift** key and clicking the last breakpoint you want to select.

The list box allows you to select multiple breakpoints that are not consecutive by clicking the first breakpoint then pressing the **Ctrl** key and clicking another breakpoint.

When multiple breakpoints are selected in the list box, the name of the group box **Breakpoint**: is changed to **Selected Breakpoints**:.

When selecting multiple breakpoints, the **Address** (hex), **Name**:, **Condition**:, **Disable** for condition, **Command**, **Current**:, and **Interval**: controls are disabled.

When multiple breakpoints are selected, the **Disable** and **Temporary** controls in the **Selected breakpoints**: group box are enabled and **Disable** in the **Command**: group box is enabled.

Checking Expressions

You can enter an expression in the **Condition:** group edit box. The syntax of the expression is checked when you select another breakpoint in the list box or click **OK**. The syntax is **parameters = = expression.** For a register condition the syntax is **$RegisterName = = expression**.

If a syntax error has been detected, a message box is displayed:

```
Incorrect Condition. Do you want to correct it?.
```

If you click **OK**, correct the error in the condition edit box.

If you click **Cancel**, the **Condition:** edit box is cleared.

Saving Breakpoints

The Debugger provides a way to store all defined breakpoints of the currently loaded application (`.ABS` file) into the matching breakpoints file. The matching file has the same name as the loaded `.ABS` file but its extension is `.BPT` (for example, the `FIBO.ABS` file has a breakpoint file called `FIBO.BPT`). This file is generated in the same directory as the `.ABS` file. This is a text file, in which a sequence of commands is stored. This file contains the following information.

- The **Save & Restore on load** flag (**Save & Restore on load** checkbox in the Controlpoints Configuration Window (Breakpoints Tab) on page 148): the **SAVEBP** command is used: **SAVEBP on** when checked, **SAVEBP off** when unchecked.

NOTE For more information about this, see the SAVEBP on page 721 command.

- List of defined breakpoints: the **BS** command is used, as shown in Listing 4.1 on page 150.

Listing 4.1 Breakpoint (.BPT) File Syntax

```
BS address [P|T[ state]][;cond="condition"[ state]]
[;cmd="command"[ state]][;cur=current[ inter=interval]]
[;cdSz=codeSize[ srSz=sourceSize]]
```

In the code above:

address is the address where the breakpoint is to be set. This address is specified in ANSI C format. **address** can also be replaced by an **expression** as shown in the example below.

P, specifies the breakpoint as a permanent breakpoint.

T, specifies the breakpoint as a temporary breakpoint. A temporary breakpoint is deleted once it is reached.

state is **E**, **D or C** where **E** is for enabled (state is set by default to **E** if nothing is specified), **D** is for disabled and **C** for Continue.

condition is an **expression.** It matches the **Condition** field in the Controlpoints Configuration Window (Breakpoints Tab) on page 148 for conditional breakpoint.

command is any debugger command. It matches the **Command** field in the Controlpoints Configuration Window (Breakpoints Tab) on page 148, for associated commands.

current is an **expression.** It matches the **Current** field (**Counter**) in the Controlpoints Configuration Window (Breakpoints Tab) on page 148, for counting breakpoints.

interval is an **expression.** It matches the **Interval** field (**Counter**) in the Controlpoints Configuration Window (Breakpoints Tab) on page 148, for counting breakpoints.

codeSize is an **expression.** It is usually a constant number to specify (for security) the code size of a function where a breakpoint is set. If the size specified does not match the size of the function currently loaded in the **.ABS** file, the breakpoint is set but it is disabled.

sourceSize is an **expression.** It is usually a constant number to specify (for security) the source (text) size of a function where a breakpoint is set. If the size specified does not match the size of the function in the source file, the breakpoint is set but it is disabled.

- If **Save & Restore on load** is checked and the user quits the Debugger or loads another .**ABS** file, all breakpoints will be saved.
- If **Save & Restore on load** is unchecked (default), only this flag (**SAVEBP off**) is saved.

Breakpoint File (.BPT) Example

Case 1: if `FIBO.ABS` is loaded, and **Save & Restore on load** was checked in a previous session of the same `.ABS` file, and breakpoints have been defined, the `FIBO.BPT` looks as shown in Listing 4.2 on page 151.

Listing 4.2 Breakpoint File with *Save & Restore on load* Checked.

```
savebp on
BS &fibo.c:Fibonacci+19 P E; cond = "fibo > 10" E; cdSz = 47 srSz = 0
BS &fibo.c:Fibonacci+31 P E; cdSz = 47 srSz = 0
BS &fibo.c:main+12 P E; cdSz = 42 srSz = 0
BS &fibo.c:main+21 P E; cond = "fiboCount==5" E; cmd = "Assembly < spc
0x800" E; cdSz = 42 srSz = 0
```

Case 2: if `FIBO.ABS` is loaded, and **Save & Restore on load was unchecked** in a previous session of the same `.ABS` file and breakpoints have been defined, the `FIBO.BPT` looks as shown below:

```
savebp on
```

Only the flag has been saved and breakpoints have been removed.

NOTE If only one or few functions differ after a recompilation, not all **BP** will be lost. To achieve that, **BP**s are disabled only if the size of a function has changed. The size of a function is evaluated in bytes (when it is compiled) and in characters (number of characters contained in the function source text). When a `.ABS` file is loaded and the matching `.BPT` file exists, for each **BS** command, the Debugger checks if the code size (in bytes) and the source size (in characters) are different in the matching function (given by the symbol table). If there is a difference, the breakpoint will be set and disabled. If there is no difference, the breakpoint will be set and enabled.

NOTE For more information about this syntax, see BS on page 659 and SAVEBP on page 721 commands.

Setting Breakpoints

The Debugger supports different types of breakpoints:

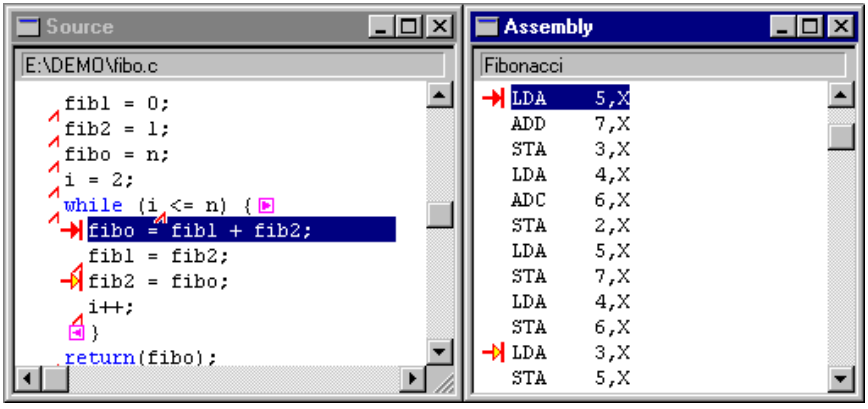
- Temporary breakpoints, which are activated next time the instruction is executed.
- Permanent breakpoints, which are activated each time the instruction is executed.
- Counting breakpoints, which are activated after the instruction has been executed a certain number of times.
- Conditional breakpoints, which are activated when a given condition is TRUE.

Breakpoints may be set in a Source or Assembly component window.

Positions Where a Breakpoint Is Definable

A compound statement is one that can be split into several base instructions. When using a high level language some compound statements can be generated, as shown in the following example.

Figure 4.3 Source and Assembly Windows



The Debugger helps you detect all positions where you can set a breakpoint.

1. Right-click in the Source component. The Source Popup Menu is displayed on the screen.
2. Choose **Marks** from the Popup Menu. All statements where a breakpoint can be set are identified by a special red inverted check mark:

To remove the breakpoint marks, right-click in the Source component and choose **Marks** again.

Temporary Breakpoints

Temporary breakpoints are activated next time the instruction is executed. A temporary breakpoint is recognized by the following icon:

Setting Temporary Breakpoints

A. Using the Source Window Popup Menu:

1. Point at a C statement in the Source window and right-click. The Source Popup Menu is displayed.
2. Choose **Run To Cursor** from the Popup Menu. The application continues execution and stops before executing the statement. You have executed a temporary breakpoint.

B, Holding down the left mouse button, pressing the T key:

1. Point at a C statement in the Source window, and holding down the left mouse button, press the T key.
2. A temporary breakpoint is defined.
3. Choose **Run To Cursor** from the Popup Menu. The application continues execution and stops before executing the statement.

Temporary breakpoints are automatically deleted once they have been activated. If you continue program execution, it will no longer stop on the statement that contained the temporary breakpoint.

Permanent Breakpoints

Permanent breakpoints are activated each time the instruction is executed. A permanent breakpoint is recognized by the following icon:

Setting Permanent Breakpoints

A. Using the Source Window Popup Menu:

1. Point at a C statement in the Source window and right-click. The Source Popup Menu is displayed.
2. Select **Set BreakPoint** from the Popup Menu. A permanent breakpoint mark is displayed in front of the selected statement.

B. Holding down the left mouse button, pressing the P key:

1. Point at a C statement in the Source window, and holding down the left mouse button, press the P key.
2. A permanent breakpoint mark is displayed in front of the selected statement.

Once a permanent breakpoint has been defined, you can continue program execution. The application stops before executing the statement. Permanent breakpoints remain active until they are disabled or deleted.

Counting Breakpoints

Counting breakpoints are activated after the instruction has been executed a certain number of times. A Counting breakpoint is recognized by the following icon:

Setting Counting Breakpoints

Counting breakpoints can only be set using the Controlpoints Configuration Window (Breakpoints Tab) on page 148. There are two ways to set a counting breakpoint:

A. Holding down the left mouse button, pressing the S key:

1. Point at a C statement in the Source window, and holding down the left mouse button, press the S key.
2. The Controlpoints Configuration window with the Breakpoints tab is opened.
3. A new breakpoint is inserted in the list of breakpoints defined in the application.
4. Select the breakpoint you want to modify by clicking on the corresponding entry in the list of defined breakpoints at the top of the tab.
5. In the **Counter:** group of this tab specify the interval for the breakpoint detection in the **Interval:** field.
6. Then close the window by clicking the **OK** button.

B. Using the Source Popup Menu:

1. Point at a C statement in the Source window and right-click. The Source Popup Menu is displayed.
2. Choose **Set BreakPoint** from the Popup Menu. A breakpoint is defined on the selected instruction.
3. Point in the Source window and right-click again.
4. Choose **Show Breakpoints** from the Popup Menu. The Controlpoints Configuration Window (Breakpoints Tab) on page 148 is displayed.
5. Select the breakpoint you want to modify by clicking on the corresponding entry in the list of defined breakpoints at the top of the tab.
6. In the **Counter:** group of this tab specify the interval for the breakpoint detection in the **Interval:** field.
7. Then close the window by clicking the **OK** button.

If you continue program execution, the content of the **Current:** field is decremented each time the instruction containing the breakpoint is reached. When **Current** is equal to 0, the application stops. If the checkbox **Temporary** is unchecked (not a temporary breakpoint),

Current is reloaded with the value stored in **Interval:** in order to enable the counting breakpoint again.

Conditional Breakpoints

Conditional breakpoints are activated when a given condition is TRUE. A conditional breakpoint is recognized by the following icon:

Setting Conditional Breakpoints

Conditional breakpoints can only be set from the Controlpoint Configuration window's Breakpoints tab. There are two ways to set a conditional breakpoint:

A. Holding down the left mouse button, pressing the S key:

1. Point at a C statement in the Source Component window, and holding down the left mouse button, press the S key.
2. The Controlpoints Configuration Window (Breakpoints Tab) on page 148 is opened and a new breakpoint is inserted in the list of breakpoints defined in the application.
3. Select the breakpoint you want to modify by clicking on the corresponding entry in the list of defined breakpoints.
4. Specify the condition for breakpoint activation in the **Condition**: group Condition box. The condition must be specified using the ANSI C syntax (Example **counter == 7**). You can use register values in the breakpoint condition field with the following syntax: **$RegisterName** (Example **$RX == 0x10**)
5. Close the window by clicking **OK**.

B. Using the Source Window Popup Menu:

1. Point at a C statement in the Source Component window and right-click. The Source Popup Menu is displayed.
2. Select **Set BreakPoint** from the Popup Menu. A breakpoint is defined on the selected instruction.
3. Point in the Source Component window and right-click. The Source Popup Menu is displayed.
4. Select Show Breakpoints from the Popup Menu. The Controlpoints Configuration Window (Breakpoints Tab) on page 148 is opened and a new breakpoint is inserted in the list of breakpoints defined in the application.
5. Select the breakpoint you want to modify by clicking on the corresponding entry in the list of defined breakpoints.

6. Specify the condition for breakpoint activation in the **Condition**: group Condition box. The condition must be specified using the ANSI C syntax (Example **counter == 7**). You can use register values in the breakpoint condition field with the following syntax: **$RegisterName** (Example **$RX == 0x10**)
7. Close the window by clicking **OK**.

If you continue program execution, the condition is evaluated each time the instruction containing the conditional breakpoint is reached. When the condition is **TRUE**, the application stops.

Deleting Breakpoints

The Debugger provides three ways to delete a breakpoint:

A. Using Delete Breakpoint from Source Popup Menu

1. In the Source component window, point at a C statement where a breakpoint has previously been defined and right-click. The Source Popup Menu is displayed.
2. Choose **Delete** Breakpoint from the Popup Menu. The breakpoint is deleted.

B. Holding down the left mouse button, pressing the D key:

1. Point at a C statement in the Source Component window where a breakpoint has previously been defined, and holding down the left mouse button, press the D key.
2. The breakpoint is deleted.

C. Choosing Show Breakpoints... from Source Popup Menu

1. Point in the Source Component window and right-click. The Source Popup Menu is displayed.
2. Choose **Show Breakpoints** from the Popup Menu. The **Breakpoints Setting** dialog is displayed.
3. In the list of defined breakpoints, select the breakpoint to delete.
4. Click **Delete**. The selected breakpoint is removed from the list of defined breakpoints.
5. Click **OK** to close the **Breakpoints Setting** dialog box.

The icon associated with the deleted breakpoint is removed from the source component.

Associate a Command with a Breakpoint

Each breakpoint (temporary, permanent, counting or conditional) can be associated with a debugger command. This command can be specified in the Breakpoints tab of the Controlpoints Configuration window. To open this window:

Choose Show Breakpoints... from Source Window Popup Menu.

1. Point in the Source Component Window and right-click. The Source Popup Menu is displayed.
2. Choose **Show Breakpoints** from the Popup Menu. The Controlpoints Configuration window with the Breakpoints tab displayed appears.

In the Breakpoints tab of the Controlpoints Configuration window:

1. You can select the breakpoint to modify by clicking on the corresponding entry in the list of defined breakpoints.
2. You can enter the command in the **Command** field. The command is a single debugger command (at this level, the commands **G**, **GO** and **STOP** are not allowed). A command file can be associated with a breakpoint using the command **CALL** or **CF** (Example: **CF breakCmd.cmd**).
3. Click **OK** to close the window.

When the breakpoint is detected, the command is executed and the application stops.

The **Continue** check button of the Controlpoints Configuration window allows the application to continue after the command is executed.

Demo Version Limitations

Only 2 breakpoints can be set.

Watchpoints

Watchpoints are control points associated with a memory range. Program execution stops when the memory range defined by the watchpoint has been accessed. The Debugger supports different types of watchpoints:

- Read Access Watchpoints, which are activated when a read access occurs inside the specified memory range.
- Write Access Watchpoints, which are activated when a write access occurs inside the specified memory range.
- Read/Write Access Watchpoints, activated when a read or write access occurs inside the specified memory range.
- Counting Watchpoints, activated after a specified number of accesses occur inside the memory range.
- Conditional Watchpoints, activated when an access occurs inside the memory range and a given condition is TRUE.

Watchpoints are controlled through the Controlpoints Configuration Window (Watchpoints Tab) on page 161. This window can be opened through the Memory or Data component window popup menu, as described below:

To open the Controlpoints Configuration window with the Watchpoints tab exposed:

1. Position your cursor in either the Memory or Data component window.
2. Press the right mouse button.
3. Select **Show Watchpoints** from either menu.
4. Click the left mouse button.

The ControlPoints Configuration window appears.The Watchpoints tab of this window is shown in Figure 4.6.

Figure 4.4 Memory Popup Menu

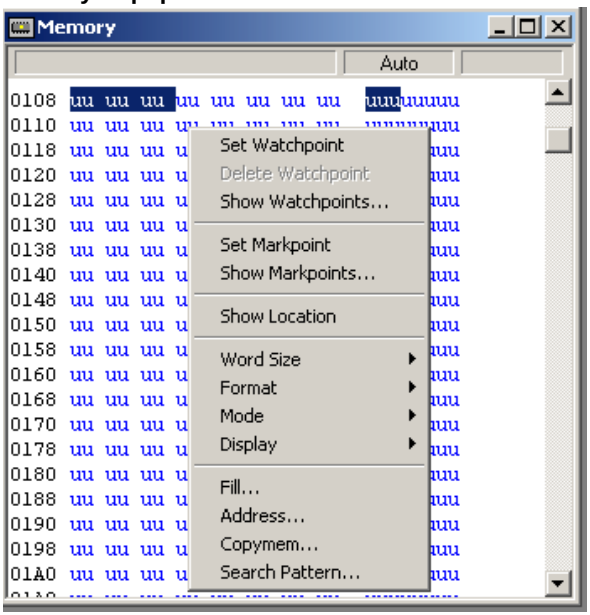

Memory
Auto
0108 uu uu uu uu uu uu uu uu uuuuuuuu
0110 uu uu uu uu uu uu uu uu uuuuuuuu
0118 uu uu uu u uuu
0120 uu uu uu u uuu
0128 uu uu uu u uuu
0130 uu uu uu u uuu
0138 uu uu uu u uuu
0140 uu uu uu u uuu
0148 uu uu uu u uuu
0150 uu uu uu u uuu
0158 uu uu uu u uuu
0160 uu uu uu u uuu
0168 uu uu uu u uuu
0170 uu uu uu u uuu
0178 uu uu uu u uuu
0180 uu uu uu u uuu
0188 uu uu uu u uuu
0190 uu uu uu u uuu
0198 uu uu uu u uuu
01A0 uu uu uu u uuu
Set Watchpoint
Delete Watchpoint
Show Watchpoints...
Set Markpoint
Show Markpoints...
Show Location
Word Size
Format
Mode
Display
Fill...
Address...
Copymem...
Search Pattern...

Figure 4.5 Data Popup Menu

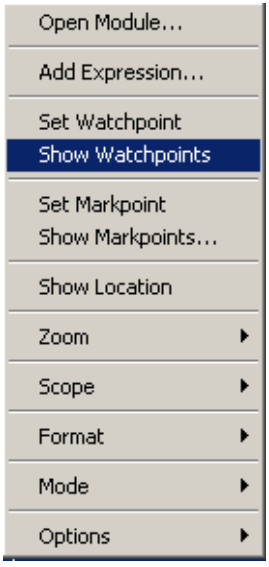
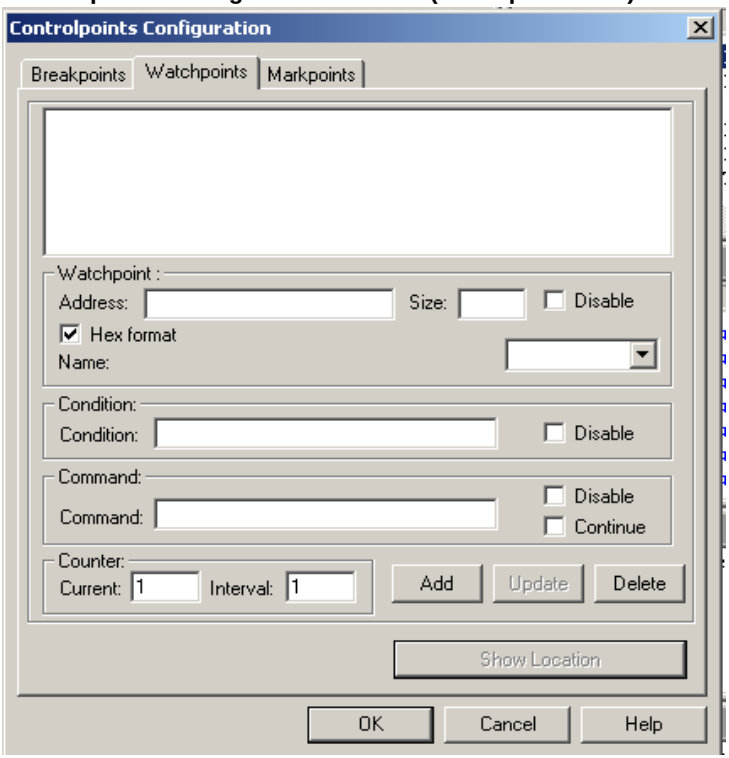


Figure 4.6 Controlpoints Configuration Window (Watchpoints Tab)



Watchpoints Tab

The Watchpoints tab of the Controlpoints Configuration window contains:

- List box that displays the list of currently defined watchpoints.
- "**Watchpoint**:" group box that displays the address of the currently selected watchpoint, size of the watchpoint, name of the procedure or variable on which the watchpoint has been set, state of the watchpoint (disabled or not), read access of the watchpoint (enabled or not) and write access of the watchpoint (enabled or not).
- "**Condition**:" group box that displays the condition string to evaluate and the state of the condition (disabled or not).
- **Update** button to Update all modifications in the dialog.
- "**Command**:" group box that displays the command string to execute and state of the command (disabled or continue after command execution).

- **Delete** button to remove currently selected watchpoint and select the watchpoint that is below the removed watchpoint.
- **OK**: button to validate all modifications.
- **Add** button to add new watchpoints; specify the Address in hexadecimal when **Hex format** is checked or as an expression when **Hex format** is unchecked.
- **Counter**: group box that displays the current value of the counter and interval value of the counter.

NOTE Current and Interval values are limited to 2,147,483,647. A beep occurs and the character is not appended, if a number greater than this value is entered.

NOTE When the Interval value is changed, the Counter value is automatically set to the Interval value.

- **Cancel** button to ignore all modifications.
- **Help**: button to display help file and related help information.

Multiple Selections

For watchpoints, you can do multiple selections in the Watchpoints tab of the Controlpoints Configuration window using the **Shift** and **Ctrl** keys.

When multiple watchpoints in the list box are selected, the name of the group box "**Watchpoint**:" is changed to "**Selected Watchpoints:**".

When multiple watchpoints are selected, the **Address** (hex), **Size**:, **Name**:, **Condition**:, **Disable** for condition, **Command**, **Current**:, and **Interval**: controls are disabled.

When multiple watchpoints are selected in the list box, the **Disable**, **Read** and **Write** controls in the **Selected watchpoints**: group box are enabled.

When multiple watchpoints are selected, **Disable** in the **Command:** group box is enabled.

Click **Delete** when multiple watchpoints are selected to remove watchpoints from the list box.

Checking Syntax

You can enter an expression in the Condition group edit box. The syntax of the expression will be checked when you select another watchpoint in the list box or by clicking **OK**.

If a syntax error has been detected, a message box is displayed:

```
"Incorrect Condition. Do you want to correct it?"
```

Click **OK** to correct the error in the condition edit box.

Click **Cancel** to clear the condition edit box.

Setting Watchpoints

Watchpoints may be set in a Data or Memory window.

NOTE Due to hardware restrictions, the watchpoint function might not be implemented on hardware connections.

Setting a Read Watchpoint

A green vertical bar is displayed in front of a variable associated with a read access watchpoint.

The Debugger provides two ways to define a read access watchpoint:

Using the Data Popup Menu:

1. Point at a variable in the Data window and right-click. The Data Popup Menu on page 160 is displayed.
2. Choose **Set Watchpoint** from the Popup Menu. A **Read/Write** Watchpoint is defined.
3. Point in the Data window and right-click. The Data Popup Menu is displayed.
4. Choose **Show WatchPoints** from the Popup Menu. The Controlpoints Configuration window Watchpoints tab is displayed.
5. Select the watchpoint you want to define as *read* access from the list.
6. Select the **Read** type in the dropdown box.
7. A read access watchpoint is defined for the selected variable.

Using the Left Mouse Button and Pressing the R Key:

1. Point at a variable in the Data window and holding down the left mouse button, press the R key.
2. A read access watchpoint is defined for the selected variable.

Once a read access watchpoint has been defined, you can continue program execution. The application stops after detecting the next read access on the variable. Read access watchpoints remain active until they are disabled or deleted.

Setting a Write Watchpoint

A red vertical bar is displayed in front of a variable associated with a write access watchpoint.

The Debugger provides two ways to define a write access watchpoint:

Using the Data Popup Menu:

1. Point at a variable in the Data window and right-click. The Data Popup Menu is displayed.
2. Choose **Set Watchpoint** from the Popup Menu. A Read/Write Watchpoint is defined.
3. Point in the Data Component Window and right-click. The Source Popup Menu is displayed.
4. Choose **Show WatchPoints** from the Popup Menu. The Controlpoints Configuration window Watchpoints tab is displayed.
5. Select the watchpoint you want to define as write access from the list.
6. Select the **Write** type in the dropdown box.
7. A write access watchpoint is defined for the selected variable.

Using the Left Mouse Button and Pressing the W Key:

1. Point at a variable in the Data window and holding down the left mouse button, press the W key.
2. A write access watchpoint is defined for the selected variable.

Once a write access watchpoint has been defined, you can continue program execution. The application stops after the next write access on the variable. Write access watchpoints remain active until they are disabled or deleted.

Defining a Read/Write Watchpoint

A yellow vertical bar is displayed in front of a variable associated with a read/write access watchpoint.

The Debugger provides two ways to define a read/write access watchpoint:

Using the Data Popup Menu:

1. Point at a variable in the Data window and right-click. The Data Popup Menu is displayed.
2. Choose **Set Watchpoint** from the Popup Menu. A Read/Write Watchpoint is defined.

Using the Left Mouse Button and Pressing the B Key:

1. Point at a variable in the Data window and holding down the left mouse button, press the B key.
2. A read/write access watchpoint is defined for the selected variable.

Once a read/write access watchpoint has been defined, you can continue program execution. The application stops after the next read or write access on the variable. Read/write access watchpoints remain active until they are disabled or deleted.

Defining a Counting Watchpoint

A counter can be associated with any type of watchpoint (read, write, read/write).

The Debugger provides two ways to define a counting watchpoint:

Using the Data Popup Menu:

1. Point at a variable in the Data window and right-click. The Data Popup Menu is displayed.
2. Choose **Set Watchpoint** from the Popup Menu. A Read/Write Watchpoint is defined.
3. Point in the Data Component Window and right-click. The Source Popup Menu is displayed.
4. Choose **Show WatchPoints** from the Popup Menu. The Controlpoints Configuration window Watchpoints tab is displayed.
5. Select the watchpoint you want to define as a counting watchpoint.
6. From the dropdown box, select the type of access you want to track.
7. In the interval field, specify the interval count for the watchpoint.
8. Close the window by clicking **OK**. A counting watchpoint is defined for the selected variable.

Using the Left Mouse Button and Pressing the S Key:

1. Point at a variable in the Data window and holding down the left mouse button, press the S key. The Watchpoints tab of the Controlpoints Configuration window is displayed.
2. Select the watchpoint you want to define as a counting watchpoint from the list.
3. From the dropdown box, select the type of access you want to track.
4. In the interval field, specify the interval count for the watchpoint. Close the window by clicking **OK**. A counting watchpoint is defined for the selected variable.

If you continue program execution, the **Current** field is decremented each time an appropriate access on the variable is detected. When **Current** is equal to 0, the application

stops. **Current** is reloaded with the value stored in the interval field to enable the counting watchpoint again.

Defining a Conditional Watchpoint

A condition can be associated with any type of watchpoint described previously (read, write, read/write).

The Debugger provides two ways to define a conditional watchpoint:

Using the Data Popup Menu:

1. Point at a variable in the Data window and right-click. The Data Popup Menu is displayed.
2. Choose **Set Watchpoint** from the Popup Menu. A Read/Write Watchpoint is defined.
3. Point in the Data window and right-click. The Source Popup Menu is displayed.
4. Choose **Show WatchPoints** from the Popup Menu. The Controlpoints Configuration window Watchpoints tab is displayed.
5. Select the watchpoint you want to define as a conditional watchpoint.
6. From the dropdown box, select the type of access you want to track.
7. Specify the condition for the watchpoint in the **Condition** field. The condition must be specified using the ANSI C syntax (Example: counter == 7).
8. Close the window by clicking **OK**. A conditional watchpoint is defined for the selected variable.

Using the Left Mouse Button and Pressing the S Key:

1. Point at a variable in the Data window and holding down the left mouse button, press the S key. The Watchpoints tab of the Controlpoints Configuration window is displayed.
2. Select the watchpoint you want to define as a conditional watchpoint.
3. From the dropdown box, select the type of access you want to track.
4. Specify the condition for watchpoint activation in the Condition field. The condition must be specified using the ANSI C syntax (Example: **counter == 7**). You can use register values in the breakpoint condition field with the following syntax: **$RegisterName** (Example **$RX == 0x10**)
5. Close the window by clicking **OK**. A conditional watchpoint is defined for the selected variable.

If you continue program execution, the condition is evaluated each time an appropriate access on the variable is detected. When the condition is TRUE, the application stops.

Deleting a Watchpoint

The Debugger provides three ways to delete a watchpoint:

Use Delete Breakpoint from Popup Menu:

1. In the Data window, point to a variable where a watchpoint has been defined and right-click. The Data Popup Menu is displayed.
2. Select **Delete Watchpoint** from the Popup Menu. The watchpoint is deleted and the vertical bar in front of the variable is removed.

Using the Left Mouse Button and Pressing the D Key:

1. Point at a variable in the Data window and holding down the left mouse button, press the D key. The Watchpoints tab of the Controlpoints Configuration window is displayed.
2. The watchpoint is deleted and the vertical bar in front of the variable is removed.

Choosing Show Watchpoints from Data Popup Menu:

1. Point in the Data window and right-click. The Data Popup Menu is displayed.
2. Choose **Show Watchpoints** from the Popup Menu. The Watchpoints tab of the Controlpoints Configuration window is displayed.
3. Select the watchpoint you want to delete.
4. Click **Delete**. The selected watchpoint is removed from the list of defined watchpoints.
5. Click **OK** to close the window. The watchpoint is deleted and the vertical bar in front of the variable is removed.

Associate a Command with a Watchpoint

Each watchpoint type (read, write, read/write, counting, or conditional) can be associated with a debugger command. This command can be specified in the Watchpoints tab of the Controlpoints Configuration window. To open this window:

Choosing Show Watchpoints... from Data Popup Menu:

1. Point in the Data Component Window and right-click. The Data Popup Menu on page 170 is displayed.
2. Select **Show Watchpoints** from the Popup Menu. The Watchpoints tab of the Controlpoints Configuration window is displayed.
3. Click on the corresponding entry in the list of defined breakpoints to select the watchpoint you want to modify.

4. You can enter the command in the **Command** field. The command is a single debugger command. At this level, the commands G on page 689, GO on page 690 and STOP on page 732 are not allowed.
   A command file can be associated with a watchpoint using the commands CALL on page 661 or CF on page 663 (Example CF breakCmd.cmd).
5. Click OK to close the window.
6. When the watchpoint is detected, the command is executed and the application will stop at this point. The **Continue** check button allows the application to continue after command execution.

Demo Version Limitations

Only two watchpoints can be set.

Markpoints

Watchpoints are control points associated with a source line, memory or data range. They provide the programmer with accessible program markers.

Program execution does NOT stop when the Source line, data or memory range defined by the markpoint has been accessed.

Markpoints are controlled through the Markpoint tab of the Controlpoints Configuration Window (Markpoints Tab) on page 171. This window can be opened through the Source, Memory or Data window popup menu, as described below:

To open the Controlpoints Configuration window with the Markpoints tab exposed:

1. Position your cursor in either the Source, Memory or Data window.
2. Press the right mouse button.
3. Select **Show Watchpoints** from the window's popup menu.
4. Click the left mouse button.

The ControlPoints Configuration window appears with the Markpoints tab of this window exposed, as shown in Figure 4.10.

Figure 4.7 Source Window Popup Menu

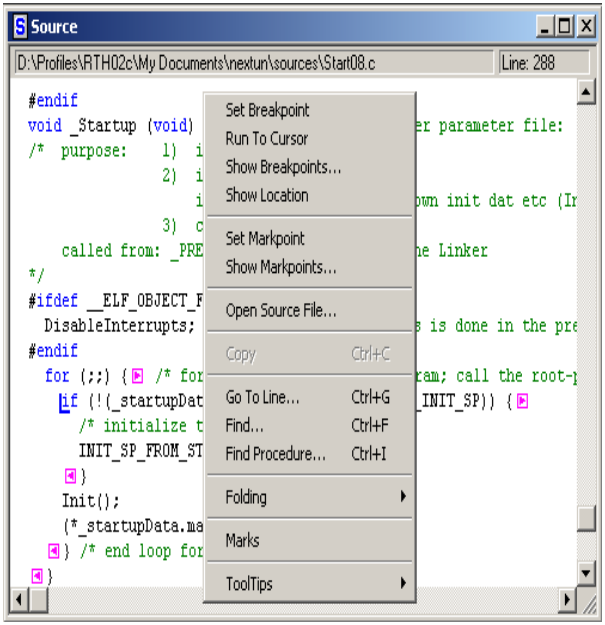


Figure 4.8 Memory Popup Menu

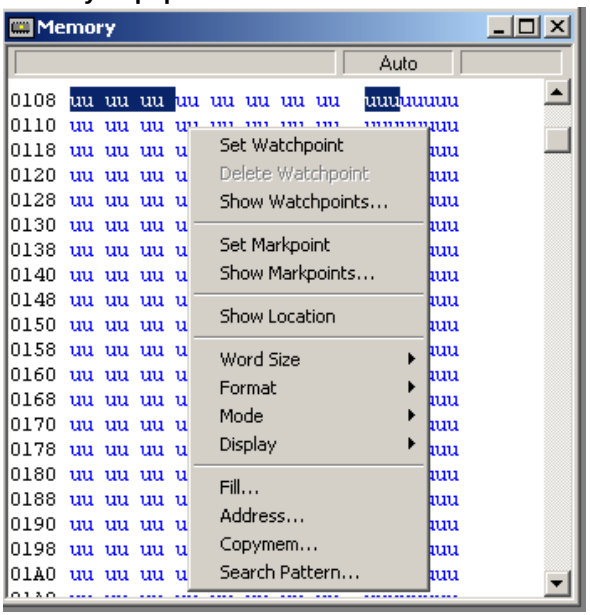
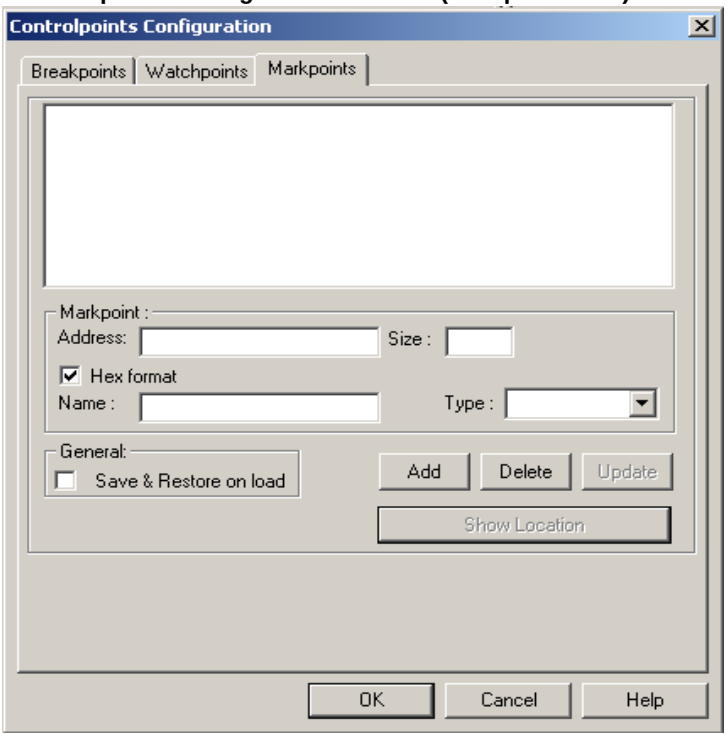

Memory
Auto
0108 uu uu uu uu uu uu uu uu uuuuuuuu
0110 uu uu uu uu uu uu uu uu uuuuuuuu
0118 uu uu uu u uuu
0120 uu uu uu u uuu
0128 uu uu uu u uuu
0130 uu uu uu u uuu
0138 uu uu uu u uuu
0140 uu uu uu u uuu
0148 uu uu uu u uuu
0150 uu uu uu u uuu
0158 uu uu uu u uuu
0160 uu uu uu u uuu
0168 uu uu uu u uuu
0170 uu uu uu u uuu
0178 uu uu uu u uuu
0180 uu uu uu u uuu
0188 uu uu uu u uuu
0190 uu uu uu u uuu
0198 uu uu uu u uuu
01A0 uu uu uu u uuu
Set Watchpoint
Delete Watchpoint
Show Watchpoints...
Set Markpoint
Show Markpoints...
Show Location
Word Size
Format
Mode
Display
Fill...
Address...
Copymem...
Search Pattern...

Figure 4.9 Data Popup Menu

Figure 4.10 Controlpoints Configuration Window (Markpoints Tab)



Markpoints Tab

The Markpoints tab of the Controlpoints Configuration window contains:

- List box that displays the list of currently defined markpoints.
- "**Markpoint**:" group box that displays the address of the currently selected markpoint, size of the markpoint, name of the procedure or variable on which the markpoint has been set, and type of the markpoint.
- "**General**" group box that contains a checkbox that allows you to save and restore the markpoint selected.
- **Add** button to add new markpoints. Specify the Address in hexadecimal when **Hex format** is checked or as an expression when **Hex format** is unchecked.
- **Delete** button to remove currently selected markpoint and select the markpoint that is below the removed markpoint.
- **Update** button to update all modifications in the window.

- **OK**: button to validate all modifications.
- **Cancel** button to ignore all modifications.
- **Help**: button to display help file and related help information.

Setting Markpoints

Markpoints may be set in a Source, Data or Memory window.

Setting a Source Markpoint

A blue letter L is displayed in front of a code line associated with a markpoint.To define a markpoint in source code:

Use the Source Popup Menu:

1. Point at a code line in the Source window and right-click. The Source Window Popup Menu on page 169 is displayed.
2. Choose **Set Markpoint** from the Popup Menu. A **markpoint** is defined at the beginning of the line.
3. Point in the Source window and right-click. The Source Popup Menu is displayed.
4. Choose **Show WatchPoints** from the popup menu. The Controlpoints Configuration Window Markpoints Tab is displayed.
5. Make any modifications to the markpoint you have installed, or any other markpoints listed.
6. Click OK to close the window.

Setting a Data Markpoint

A blue letter L is displayed in front of a variable associated with a markpoint. To define a data range markpoint:

Use the Data Popup Menu:

1. Point at a variable in the Data window and right-click. The Data Popup Menu on page 170 is displayed.
2. Choose **Set Markpoint** from the popup menu. A markpoint is defined at the beginning of the data range selected.
3. Point in the Data window and right-click. The Data Popup Menu is displayed.
4. Choose **Show WatchPoints** from the popup menu. The Controlpoints Configuration Window Markpoints Tab is displayed.

5. Make any modifications to the markpoint you have installed, or any other markpoints listed.
6. Click OK to close the window.

Setting a Memory Markpoint

A blue letter L is displayed in front of a memory range associated with a markpoint.

To define a Memory markpoint:

Use the Memory Popup Menu:

1. Point at a line in the Memory window and right-click. The Memory Popup Menu on page 170 is displayed.
2. Choose **Set Watchpoint** from the Popup Menu. A Markpoint is defined.
3. Point in the Memory window and right-click. The Memory Popup Menu is displayed.
4. Choose **Show WatchPoints** from the Popup Menu. The Controlpoints Configuration Window Markpoints Tab is displayed.
5. Make any modifications to the markpoint you have installed, or any other markpoints listed
6. Click OK to close the window.

Deleting a Markpoint

To delete a markpoint:

Using the Left Mouse Button and Pressing the D Key:

1. Point at the markpointed variable in the Data window, the memory range in the Memory window, or the codeline in the Source window:
2. Holding down the left mouse button, press the D key.
3. The markpoint is deleted and the blue letter L in front of the variable, memory range or codeline is removed.

Choosing Show Markpoints from Appropriate Popup Menu:

1. Point in the Data, Memory or Source component window and right-click. That window's popup menu is displayed.
2. Choose **Show Markpoints** from the Popup Menu. The Markpoints Tab of the Controlpoints Configuration Window is displayed.
3. In this tab's List box, select the markpoint(s) you want to delete.
4. Click **Delete**. The selected markpoint is removed from the list of defined watchpoints.

5. Click **OK** to close the window. The markpoint is deleted and the blue letter L in front of the variable, memory range, or code line is removed.

Halting on a Control Point

Code execution is halted when the program reaches either a breakpoint or a watchpoint, if the conditions specified in the definition of the breakpoint or watchpoint have been reached. Code execution is NOT halted when the program reaches a markpoint.

Counting Control Point

If the interval property is greater than 1, a counting control point has been defined. When the Debugger is running, each time the control point is reached, its current value is decremented and the Debugger will halt when the value reaches zero (0). When the Debugger stops on the control point, a command will be executed (if defined and enabled).

Conditional Control Point

If a condition has been defined and enabled for a control point that halts the Debugger, a command will be executed (if defined and enabled).

Control Point with Command

When the Debugger halts on the control point, a specified command is executed.

5

Real Time Kernel Awareness

The Debugger allows you to load and control applications on the target system, or applications simulated on the host. It also allows you to inspect the state of the application, which includes global variables, processor registers and the procedure call chain including the local (automatic) variables.

This chapter describes how applications built of several tasks are handled by a generic awareness support and an OSEK awareness.

Click any of the following links to jump to the corresponding section of this chapter:

- Introduction on page 175
- Task Description Language on page 176
- Application Example on page 178
- Inspecting Kernel Data Structures on page 179
- OSEK Kernel Awareness on page 180

Introduction

Often operating systems (Real Time Kernels) are used to coordinate the different tasks in more complex systems. This chapter describes how applications built of several tasks can be handled with the Debugger. There are two main topics to be considered:

- Debugging of any task in the system (e.g., viewing the state of any task in the system). When using the original basic versions of the Debugger, only the current task can be inspected. Due to this extension, it is possible to switch the debugging context from the current task to any other task and between any tasks in the system.
- Real time kernels use data structures to describe the state of the system (scheduling information, queues, timers,...). Some of these data structures are interesting for the user of an operating system too and are described in this chapter.

Inspecting Task State

Each multitasking operating system stores the context of each task at a specific location, usually called the task descriptor. This context consists of the CPU context (CPU registers) and the content of the associated stack. There is more information in the task descriptor, depending on the specific implementation of the kernel.

The Debugger allows you to inspect the CPU registers and stack containing all procedure activation frames (return addresses, parameters, local variables). Therefore, it has to get this information for each task to be debugged. Since this information is specific to the kernel used, there is a universal way to specify the location where and how to collect this data.

This information is read from a file with the name 'OSPARAM.PRM', which describes the algorithm of how to get all the needed data from the target memory (from the task descriptors). To describe this algorithm, a simple procedural language is used. The only parameter to the algorithm is an address specified by the user, which identifies the task to be inspected. The result is the CPU context (CPU registers) and status of the task, which allows the debugger to display the procedure activation stack in a symbolic way.

RTK Interface

When the application is halted, the debugger displays the state of the current task. To identify the task to be inspected, the user has to follow these steps.

Make the task descriptor or a pointer to it visible in any of the debugger's data windows.

Press the **P** key while holding down the left mouse button on a variable of type "pointer to task descriptor".

Now the current state of the selected task and procedure chain of that task is displayed in the 'Procedure Chain' window. By clicking on the procedures in the call chain list, the local data of that function is displayed in the 'Data1' window. All the usual debugging functions are also available to inspect this task now (including displaying the register contents).

Task Description Language

To perform debugging on any task, a file named "OSPARAM.PRM" has to be created and must be stored in one of the directories specified in GENPATH: #include “File” Path on page 791

The file "OSPARAM.PRM" describes the algorithm to collect the context information for a specific task (the PC, SP, DL, SR and registers).

The following syntax has to be used to specify the algorithm (in EBNF):

```
StatSequence          =     [Statement] {';' Statement;}.
Statement             =     Assignment | ErrorMsg | If.
Assignment            =     Ident ':=' Expression.
ErrorMsg              =     'MSG' ':=' String.
IfStatemen            =     'IF' BoolExpr 'THEN' StatSequence
                      {ELSIFPart} [ELSEPart] 'END'.
ELSIFPart             =     'ELSIF' BoolExpr 'THEN' StatSequence.
ELSEPart              =     'ELSE' StatSequence.
String                =     '"' {char} '"'.
BoolExpr              =     Expression RelOp Expression.
Expression            =     Term {Op Term}.
Term                  =     Ident | Function | Number.
Ident                 =     'a'..'z' | 'R00'..'R31' | 'DL' | 'SP' |
                      'SR' | 'PC' | 'STATUS' | 'B'.
Function              =     ('MB' | 'MW' | 'MD' | 'MA') '['
                      Expression ']'.
RelOp                 =     '#' | '<' | '<=' | '=' | '>=' | '>'.
Op                    =     '+' | '-'.
```

The terminal symbols have the following meaning:

B	is the given reference to the task descriptor (initialized upon start).
a..z	are variables for intermediate storage.
MB	gets value of memory BYTE at given address.
MW	gets value of memory WORD at given address.
MD	gets value of DOUBLE WORD at given address.
MA	gets value at given address interpreted as DOUBLE WORD.
PC	is the program counter to be set.
SP	is the stack pointer to be set.
SR	is the status register value to be set.
DL	is the dynamic link (data base) to be set (if not available, same as SP).
STATUS	is the error number to be set (refer to manual).
Rnn	processor registers to be set (mapping to CPU registers see manual).
MSG	is the error message (has to be specified if N >= 1000).

On activation of the task debugging command, the file `"OSPARAM.PRM"` is opened and the selected address is stored in variable **'B'**. Then the commands in the file are interpreted. The CPU context of the task is then expected in the variables PC, SP, SR, DL,

Rnn and EN. EN describes the status of the task. If 'EN' is bigger than 1000 the status is expected in the string MSG.

Application Example

Listing 5.1 on page 178 shows an example of *"OSPARAM.PRM"* file for SOOM System/REM.

Listing 5.1 OSPARAM.PRM File

```
{ File OSParam.PRM, implementation for SOOM System/REM }
{ R0..R7 = D0..D7, R8..R15 = A0..A7 }
{ MSG = message displayed in Procedure Chain window }

DL := MD(B+8);  { A6 in PD, dynamic link    }
SP := MD(B+4);  { A7 in PD, stack pointer   }
PC := MD(B+14); { PC in PD, program counter }
SR := MW(B+12); { SR in PD, status register }
STATUS := 1000; { Initialized with 1000 }
IF MW(B+18) = 1 THEN
{ IF (registers are saved in task Control Block) THEN }
R0 := MD(B+22); R1 := MD(B+26); R2 := MD(B+30);
R3 := MD(B+34); R4 := MD(B+38); R5 := MD(B+42);
R6 := MD(B+46); R7 := MD(B+50); R8 := MD(B+54);
R9 := MD(B+58); R10 := MD(B+62); R11 := MD(B+66);
R12 := MD(B+70)
END;
R13 := B;
R14 := DL;
R15 := SP;
i := MB(B+112); { i contains the current state of the selected task. }
IF i = 0 THEN MSG := "ReadyInCQSc"
ELSIF i = 1 THEN MSG := "BlockedByAccept"
ELSIF i = 2 THEN MSG := "WaitForDReply"
ELSIF i = 3 THEN MSG := "WaitForMail"
ELSIF i = 4 THEN MSG := "DelayQueue"
ELSIF i = 5 THEN MSG := "BlockedByReceive"
ELSIF i = 6 THEN MSG := "WaitForSemaphore"
ELSIF i = 7 THEN MSG := "Dummy"
ELSIF i = 8 THEN MSG := "SysBlocked"
ELSE  MSG := "invalid"
END;
```

Inspecting Kernel Data Structures

To allow the debugger to display the data structures of the operating system, the corresponding symbol information has to be available. This is the case when using SOOM System/REM. When another kernel is used its source code would have to be available and would have to be compiled. However, if only the object code is available, the needed symbol information can be generated in the following way:

- The kernel data structures of interest have to be described using ANSI-C language, as shown in Listing 5.2 on page 179.

Listing 5.2 Kernel Data Structure Description

```
typedef struct PD {
   int status;
   struct PD *next;
   long regs[6];
} PD;
```

This is an example of the definition of a simple task descriptor.

- Variables can be collected in a structure and have to be assigned to a segment (for example, 'OS_DATA' shown in Listing 5.3 on page 179).

Listing 5.3 OS_DATA Structure

```
#pragma DATA_SEG OS_DATA
struct {
  PD *readyList;     /* list of tasks ready to be executed */
  char filler[6];    /* unimportant variables */
  int processes;     /* total number of tasks */
  PD processes[10]; /* the 10 possible tasks */
} OS_DATA;
```

This structure should be defined so as to fit the same layout as the operating system used. It might be necessary to introduce filler variables to get the correct alignment.

This segment has to be placed by the linker to the correct address by using the PRM file shown in Listing 5.4 on page 179:

Listing 5.4 Linker PRM File

```
NAMES  ... rtk.o+ ... END
SECTIONS
   ...
   RTK_SEC = NO_INIT 0x1040 TO 0x1F80;
   ...
```

```
END

PLACEMENT
   ...
   OS_DATA INTO RTK_SEC;
   ...
END
```

The source file (for example: 'rtk.c') has to be compiled and listed in the NAMES section of the linker parameter file. To force linking, the name of the object file has to be immediately followed by a '+'. In this example the variable is linked to the address 0x1040.

If an application is prepared in this way, all declared variables may be inspected in the data windows of the Debugger. There is no restriction in the complexity of the structures to describe the global data of the kernel.

NOTE You should not open the terminal window during testing. Errors detected during reading of a PRM file are written to this window.

OSEK Kernel Awareness

OSEK Kernel provides a framework for building real-time applications.

OSEK Kernel awareness within the debugger allows you to debug your application from the operating system perspective.

The CodeWarrior Debugger supports OSEK ORTI compliant real-time operating systems and offers dedicated kernel awareness, using the information stored in your application's ORTI file.

With CodeWarrior OSEK kernel awareness, you can monitor kernel task information, semaphores, messages, queues, resources allocations, synchronization, communicating between tasks, etc.

ORTI describes the applications in any OSEK implementation:

- A set of attributes for system objects.
- A method for interpreting the data obtained.

OSEK ORTI

The OSEK Run Time Interface (ORTI) is an interface for development tools to the OSEK Operating System. It is a part of the OSEK standard (refer to www.osek-vdx.org).

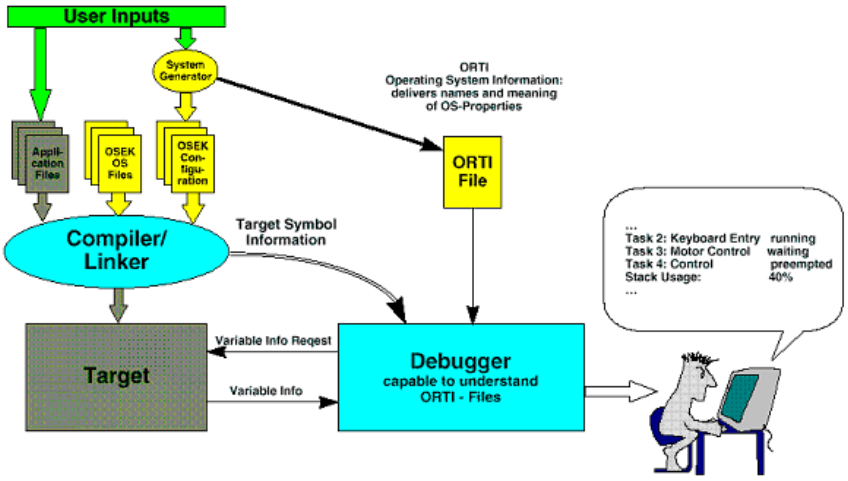
The ORTI enables the attached tool to evaluate and display information about the operating system, its state, its performance, the different task states, the different operating system objects etc.

ORTI File and Filename

The ORTI file name has the same name as the application file name, but with the extension .ort. For instance, if the application file name is `winLift_demo.abs`, the ORTI file name is `winLift_demo.ort`. Otherwise the debugger cannot use the correct ORTI file.

The ORTI file contains dynamic information as a set of attributes that are represented by formulas to access corresponding dynamic values. Formulas for dynamic data access are comprised of constants, operations, and symbolic names within the target file. The given formula can then be evaluated by the debug tool to obtain internal values of the required OS objects.

Figure 5.1 ORTI Aware Debugging System



Two types of data are made available to the CodeWarrior debug tool. One type describes static configuration data that remains unchanged during program execution. The second type of data is dynamic and this data is re-evaluated each time by CodeWarrior. The static information is useful for display of general information and in combination with the dynamic data. The dynamic data gives information about the current status of the system.

The information given to CodeWarrior is represented in a text (ORTI-File). The file describes the different objects configured in the OS and their properties. The information is represented in direct text, enumerated values, Symbolic names, or an equation that may be used for evaluating the attribute.

The ORTI File is generated when building the project through the OSEK System Generator. The generated file has the same name and the same location as executable file but its extension is .ort.

ORTI File Structure

The ORTI file structure builds on top of the structure of the OSEK OIL file. It consists of the following parts:

- Version Section - This section describes the version of the ORTI standard used for the current ORTI file.
- Implementation Definition Section - This section describes the method that should be used to interpret the data obtained for the value. This section may also detail the suggested display name for a given attribute.
- Application Definition Section - This section contains information on all objects that are currently available for a given system. This section also describes the method that shall be used to reference or calculate each required attribute. This information shall either be supplied as a static value or else a formula that shall be used to calculate the required value.

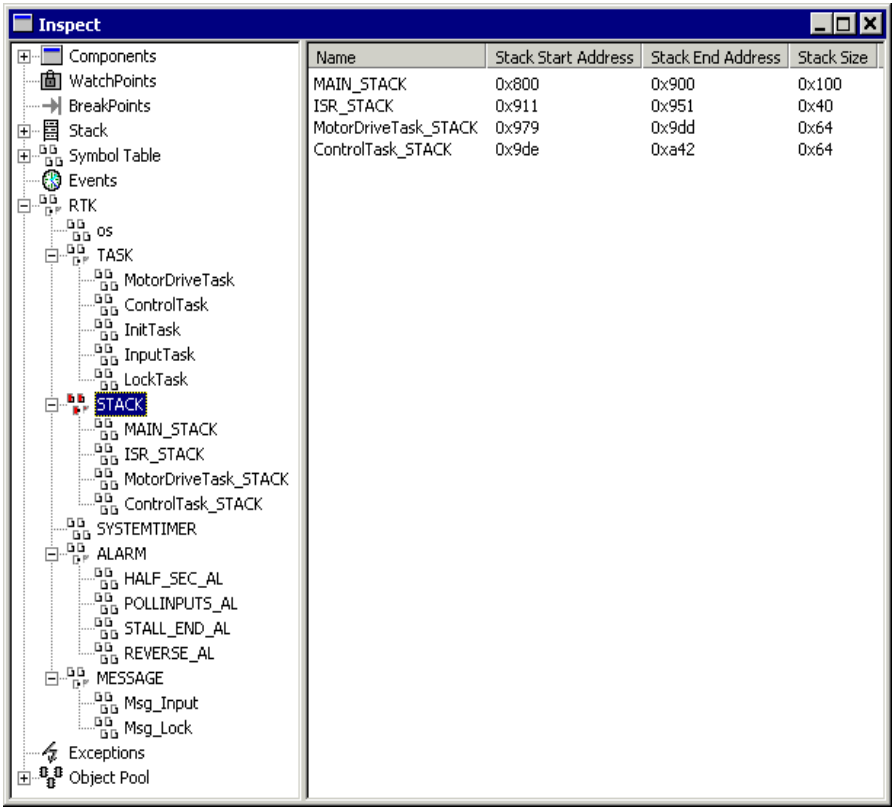
OSEK RTK Inspector Component

OSEK awareness is described through the Code Warrior RTK Inspector component as shown in Figure 5.2 on page 183.

Inspector window is displayed by clicking on ***Component>Open...*** menu entry and then by clicking on ***Inspect*** icon in the “Open Window Component” window.

When the RTK components icon is selected in the hierarchical content of the items, the right side displays various information about OSEK Awareness.

Figure 5.2 CodeWarrior RTK Inspect Window



The OSEK RTK Inspect Window provides access to all this information. As defined in the ORTI file, objects of the same type are grouped and can be viewed together.

- Task
- Stack
- SystemTimer
- Alarm
- Message.

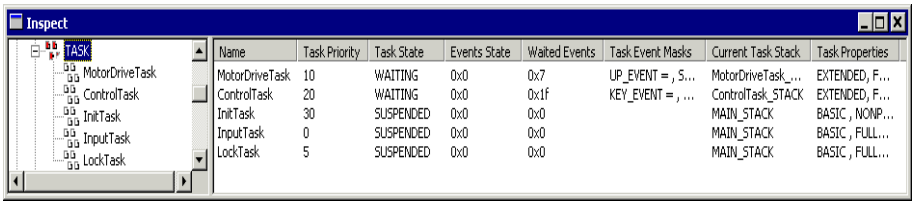
The following sections offer a description of typical objects along with their attributes and how they are presented.

NOTE Objects and their attributes depend on the OSEK implementation and OSEK configuration, and therefore may differ from this description.

Inspector Task

The Task shown in Figure 5.3 on page 184 displays the current state of the OSEK task trace.

Figure 5.3 Inspector Task



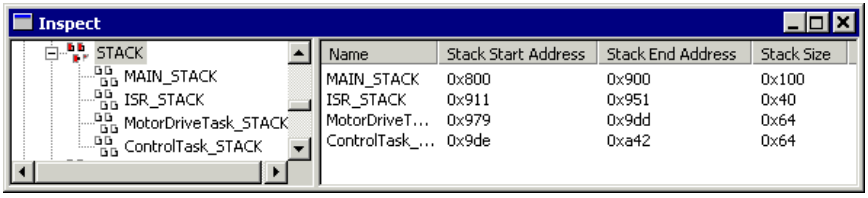
When selecting Task in the hierarchical tree on the left side of the Inspect window, additional information concerning tasks is displayed on the right side of the window under the following headings:

- **Name**: displays the name of the task
- **Task Priority**: displays the priority of the task.
- **Task State**: describes the current state of the task. Possible values are READY, SUPENDED, WAITING, RUNNING or INVALID_TASK. The ORTI file defines the different states.
- **Events State**: the event is represented by its mask. The event mask is the number which range is from 1 to 0xFFFFFFFF. When the event mask value is set to1, the event is activated. When it is set to 0, the event is disabled.
- **Waited Events**: when the bit is set to 0, the event is not expected. When the bit is set to 1, the event is expected.
- **Task Event Masks**: describes the current task event mask.
- **Current Task Stack**: displays the name of the current stack used by the task.
- **Task Properties**: describes task properties. Possible value are BASIC/EXTENDED, NONPREMPT/FULLPREMPT, Priority value, AUTO. The ORTI file defines the possible values.

Inspector Stack

The Stack shown in Figure 5.4 on page 185 displays the current state of OSEK stack trace.

Figure 5.4 Inspector Stack



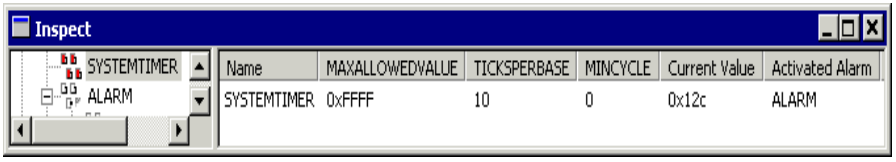
When selecting Stack in the hierarchical tree on the left side, additional information concerning the stack are displayed on the right side of the window under the following headings:

- **Name**: displays the name of the stack.
- **Stack Start Address**: displays the start address of the stack.
- **Stack End Address**: displays the end address of the stack.
- **Stack Size**: displays the size of the stack.

Inspector SystemTimer

The SystemTimer shown in Figure 5.5 on page 185 displays the current state of OSEK SystemTimer trace.

Figure 5.5 Inspector SystemTimer



When selecting SystemTimer in the hierarchical tree on the left side, additional information concerning the timer are displayed on the right side of the window under the following headings:

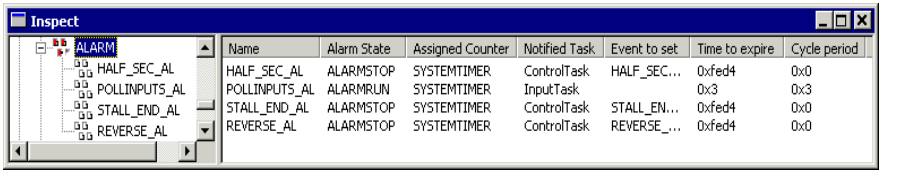
- **Name**: displays name of the system timer.
- **MAXALLOWEDVALUE**: displays the maximum allowed counter value. When the counter reaches this value it rolls over and starts count again from zero.
- **TICKSPERBASE:** displays the number of ticks required to reach a counter-specific value.

- **MINCYCLE**: displays the minimum allowed number of counter ticks for a cyclic alarm linked to the counter.
- **Current Value**: displays the current value of the system timer.
- **Activated Alarm**: displays associated alarms.

Inspector Alarm

The Alarm shown in Figure 5.6 on page 186 displays the current state of OSEK alarm trace.

Figure 5.6 Inspector Alarm



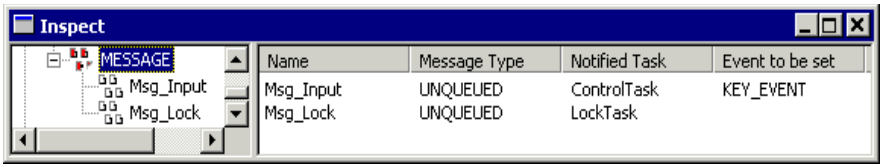
When selecting Alarm in the hierarchical tree on the left side, additional information concerning the alarm are displayed on the right side of the window under the following headings:

- **Name**: displays the name of the alarm.
- **Alarm State**: displays the current state of the alarm. Possible values are ALARMRUN and ALARMSTOP.
- **Assigned Counter**: based on counters, the OSEK OS offers alarm mechanism to the application software. Assigned Counter is the name of the counter used by alarm.
- **Notified Task**: alarm management allows the user to link task activation to a certain counter value, the assignment of an alarm to a counter, as well as the action to be performed when an alarm expires. Notified Task defines the task to be notified (by activation or event setting) when the alarm expires.
- **Event to Set**: alarm management allows the user to link event setting to a certain counter value, the assignment of an alarm to a counter, as well as the action to be performed when an alarm expires. Event to set specifies the event mask to be set when the alarm expires.
- **Time to expire**: displays time remaining before the time expires and the event is set.
- **Cycle period**: displays period of a tick.

Inspector Message

The Message shown in Figure 5.7 on page 187 displays the current state of OSEK message trace.

Figure 5.7 Inspector Message



When selecting Message in the hierarchical tree on the left side, additional information concerning task are displayed on the right side:

- **Name**: displays the name of the message.
- **Message Type**: displays message type. Possible values are: UNQUEUED/ QUEUED.
- **Notified Task**: displays the task that shall be activated when the message is sent.
- **Event to be set**: displays the event which is to be set when the message is sent.

6

How To ...

This chapter provides answers to frequently asked questions. Click any of the following links to jump to the corresponding set of instructions:

- How To Configure the Debugger on page 190
- How To Start the Debugger on page 191
- Automating Debugger Startup on page 192
- How To Load an Application on page 193
- How To Start an Application on page 194
- How To Stop an Application on page 194
- How To Step in the Application on page 195
- How To Work on Variables on page 197
- How To Work on the Register on page 201
- Modify Content of Memory Address on page 203
- How to Consult Assembler Instructions Generated by a Source Statement on page 203
- How To View Code on page 204
- How to Communicate with the Application on page 205
- About startup.cmd, reset.cmd, preload.cmd, postload.cmd on page 205

How To Configure the Debugger

If you have installed the Debugger under Windows 95, 98, NT 4.0 and Windows2000 or higher, the Debugger can be started from the CodeWarrior IDE, from the desktop, from the Start menu, or from an external editor (WinEdit, CodeWright, etc.). In order to work efficiently (find all requested configuration and component files), the Debugger must be associated with a working directory.

For Use from Desktop (Win 95, Win 98,Win NT4.0 or Win2000)

When starting the Debugger from Windows 95 or Windows NT V4.0 (for example, without WinEdit), the working directory can be defined in the file **MCUTOOLS.INI**, located in the Windows directory.

Defining the Default Directory in the MCUTOOLS.INI

When starting from the desktop or Start menu, the working directory can be set in the configuration file `MCUTOOLS.INI`.

The working directory including the path is defined in the environment variable **DefaultDir** in the [**Options**] group or **WorkDir** [**WorkingDirectory**].

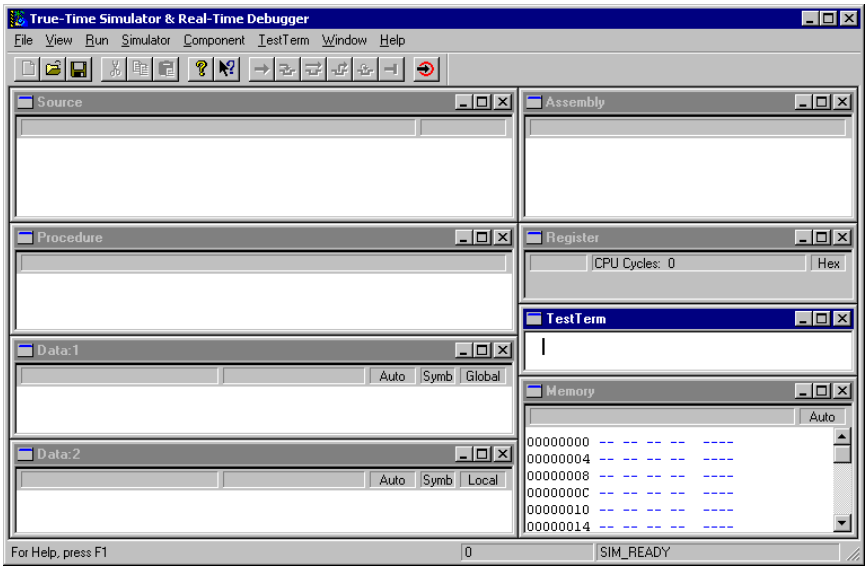
How To Start the Debugger

This section describes various ways to start the Debugger.

From WinEdit

The Debugger can be started by selecting **Project>Debug** or clicking the Debugger icon (bug) in WinEdit tool bar (when configured). The Window looks like Figure 6.1 on page 191.

Figure 6.1 Debugger After Startup



READY displayed in the status bar indicates that the simulator is ready.

Automating Debugger Startup

Often the same tasks have to be performed after starting the Debugger. These tasks can be automated by writing a command file that contains all commands to be executed after startup of the Debugger, as shown in Listing 6.1 on page 192.

Listing 6.1 Example of a Command File to Automate Tasks

```
load fibo.abs
bs &main t
g
```

This file will first load an application, then set a temporary breakpoint at the start of the function **main** and start the application. The application will then stop on entering **main** (after executing the startup and initialization code).

There are several ways to execute this command file:

- specify the command file on the command line using the command line option **-c**: This is done in the application that starts the Debugger (for example, Editor, Explorer, Make utility, ...).

Example:

```
\Freescale\PROG\HIWAVE.EXE -c init.cmd
```

When the Debugger is started with this command line, it will execute the command specified in the file `init.cmd` after loading the layout (or project file).

- Calling the command file from the project file (Listing 6.2 on page 192). The project file where the layout and connection component can be saved (**File >Save...)** is a normal text file that contains command line commands to restore the context of a project. This file, once created by the save command, can be extended by a call to the command file (**CALL INIT.CMD**). When this project is loaded by the **File >Open...** command or by the corresponding entry in the Project file, commands in this file are executed.

Listing 6.2 Calling a Command File from the Project File:

```
set Sim
CLOSE *
call \Freescale\DEMO\test.hwl
call init.cmd
```

- Calling the command file when the Connection Component is loaded. Most connection components will execute the command file `STARTUP.CMD` once the connection component is loaded and initialized. By adding the call command file in

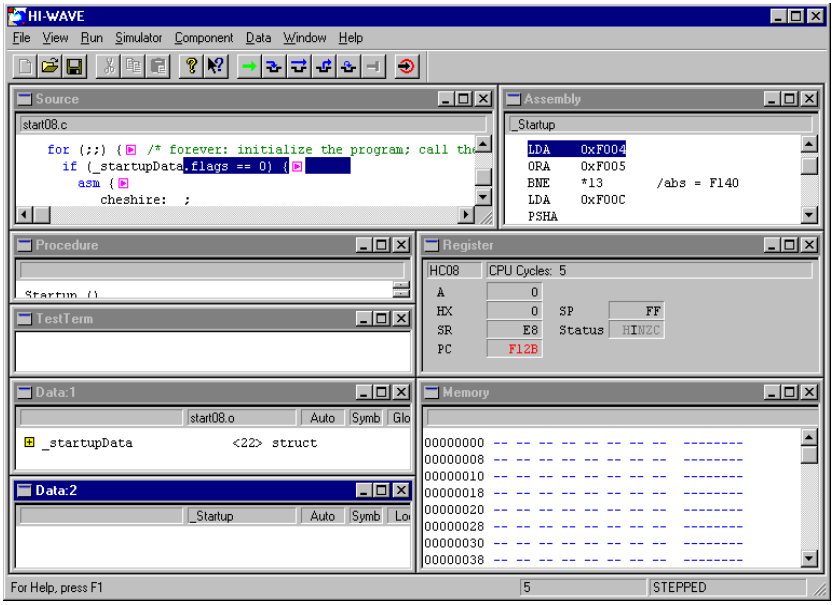
this file (for example, **CALL INIT.CMD**), it will automatically execute when the connection component is loaded.

NOTE Refer to section How To Start the Debugger on page 191.

How To Load an Application

1. Choose **Simulator > Load** .... The **LoadObjectFile** dialog box is opened.
2. Select an application (for example FIBO.ABS).
3. Click **OK**. The dialog box is closed and the application is loaded in the Debugger (Listing 6.2 on page 192).

Figure 6.2 Load an Application in the Debugger.


HI-WAVE
File View Run Simulator Component Data Window Help
Source
start08.c
for (;;) { /* forever: initialize the program; call the
if (_startupData.flags == 0) {
asm {
cheshire: ;
Assembly
_Startup
LDA 0xF004
ORA 0xF005
BNE *13 /abs = F140
LDA 0xF00C
PSHA
Procedure
Register
HC08 CPU Cycles: 5
A 0
HX 0 SP FF
SR E8 Status HINZC
PC F12B
TestTerm
Data:1
start08.o Auto Symb Glo
_startupData <22> struct
Data:2
_Startup Auto Symb Lo
Memory
00000000 -- -- -- -- -- -- -- -- --------
00000008 -- -- -- -- -- -- -- -- --------
00000010 -- -- -- -- -- -- -- -- --------
00000018 -- -- -- -- -- -- -- -- --------
00000020 -- -- -- -- -- -- -- -- --------
00000028 -- -- -- -- -- -- -- -- --------
00000030 -- -- -- -- -- -- -- -- --------
00000038 -- -- -- -- -- -- -- -- --------
For Help, press F1
5
STEPPED

The Source component contains source from the module containing the entry point for the application (usually the startup module). The highlighted statement is the entry point.

The Assembly component contains the corresponding disassembled code. The highlighted statement is the entry point.

The Global Data component contains the list of global variables defined in the module containing the application entry point.

The Local Data component is empty.

The PC in the Register component is initialized with the PC value from the application entry point.

How To Start an Application

There are two different ways to start an application:

1. Choose **Run>Start/Continue**

2. Click the **Start>Continue** icon in the debugger tool bar

RUNNING in the status line indicates that the application is running.

The application will continue execution until:

- you decide to stop the execution (See How To Stop an Application on page 194).
- a breakpoint or watchpoint has been reached.
- an exception has been detected (watchpoints or breakpoints).

How To Stop an Application

There are two different ways to stop program execution:

1. Choose **Run >Halt**

2. Click on the **Halt** icon in the debugger tool bar

HALTED in the status line indicates that execution has been stopped.

The blue highlighted line in the source component is the source statement at which the program was stopped (next statement to be executed).

The blue highlighted line in the Assembly component is the assembler statement at which the program was stopped (next assembler instruction to be executed).

Data window with attribute **Global** displays the name and values of the global variables defined in the module where the currently executed procedure is implemented. The name of the module is specified in the Data info bar.

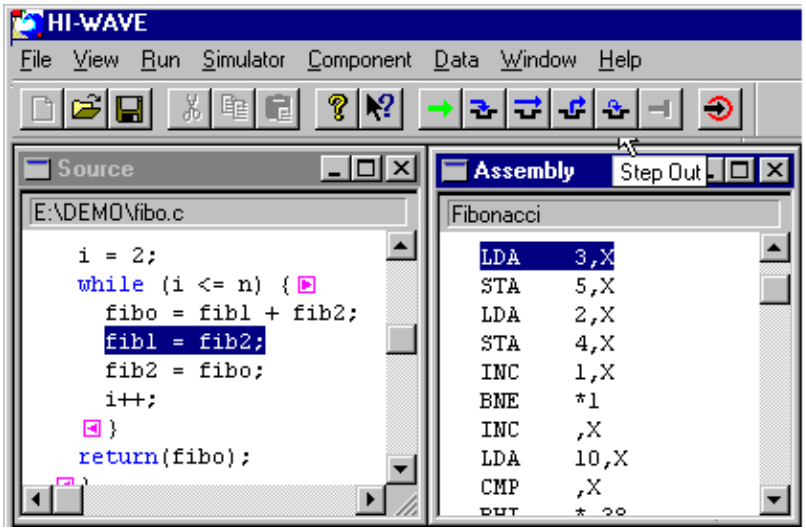
Data window with attribute **Local** displays the name and values of the local variables defined in the current procedure. The name of the procedure is specified in the Data info bar.

How To Step in the Application

The Debugger provides stepping functions at the application source level and assembler level (Figure 6.3 on page 195).

On Source Level

Figure 6.3 Stepping on Source Level.



On the Next Source Instruction

The Debugger provides two ways of stepping to the next source instruction:

1. Choose **Run>Single Step**
2. Click the **Single Step** icon from the Debugger tool bar
3. STEPPED in the status line indicates that the application is stopped by a step function.

If the application was previously stopped on a subroutine call instruction, a **Single Step** stops the application at the beginning of the invoked function.

The display in the Assembly component is always synchronized with the display in the Source component. The highlighted instruction in the Assembly component is the first assembler instruction generated by the highlighted instruction in the Source component.

Elements from Register, Memory or Data components that are displayed in red are the register, memory position, local or global variables, and which values have changed during execution of the source statement.

Step Over a Function Call (Flat Step)

The Debugger provides two ways of stepping over a function call:

1. Choose **Run >Step Over**

2. Click the **Step Over** icon from the Debugger tool bar

STEPPED OVER (STEPOVER on page 731) or STOPPED (STOP on page 732) in the status line indicates that the application is stopped by a step over function.

If the application was previously stopped on a function invocation, a **Step Over** stops the application on the source instruction following the function invocation.

The display in the Assembly component is always synchronized with the display in the Source component. The highlighted instruction in the Assembly component is the first assembler instruction generated by the highlighted instruction in the Source component.

Elements from Register, Memory or Data components that are displayed in red are the register, memory position, local or global variables, and which values have changed during execution of the invoked function.

Step Out from a Function Call

The Debugger provides two ways of stepping out from a function call:

1. Choose Run>Step Out

2. Click the **Step Out** icon from the debugger tool bar

STOPPED (STOP on page 732) in the status line indicates that the application is stopped by a step out function.

If the application was previously stopped in a function, a **Step Out** stops the application on the source instruction following the function invocation.

The display in the Assembly component is always synchronized with the display in the Source component. The highlighted instruction in the Assembly component is the first assembler instruction generated by the highlighted instruction in the Source component.

Elements from Register, Memory or Data components that are displayed in red are the register, memory position, local or global variables, and which values have changed since the **Step Out** was executed.

Step on Assembly Level

The Debugger provides two ways of stepping to the next assembler instruction:

1. Choose **Run>Assembly Step**

2. Click the **Assembly Step** icon from the debugger tool bar

TRACED in the status line indicates that the application is stopped by an assembly step function.

The application stops at the next assembler instruction.

The display in the Source component is always synchronized with the display in the Assembly component. The highlighted instruction in the Source Component is the source instruction that has generated the highlighted instruction in the Assembly component.

Elements from Register, Memory or Data components that are displayed in red are the register, memory position, local or global variables, and which values have changed during execution of the assembler instruction.

How To Work on Variables

This section shows the different methods to work on variables.

Display Local Variable from a Function

The Debugger provides two different ways to see the list of local variables defined in a function:

- Using Drag and Drop

1. Drag a function name from the Procedure component to a Data component with attribute **local**.

- Using Double-click

1. Double-click a function name in the Procedure component.

The Data component (with attribute **local** that is neither **frozen** or **locked)** displays the list of variables defined in the selected function with their values and type.

Display Global Variable from a Module

The Debugger provides two ways to see a list of global variables defined in a module:

- Opening Module Component

1. Choose **Component>Open**. The list of all available components is displayed on the screen.
2. Double-click the entry **Module**. A module component is opened, which contains the list of all modules building the application.
3. Drag a module name from the **Module** component to a Data component with attribute **Global**.

- Using Popup Menu

1. Right-click in a Data component with attribute **Global**.
2. Choose **Open Module** in Popup Menu. A dialog box is opened, which contains the list of all modules building the application.

- Double-click on a module name. The Data component with attribute **global**, which is neither **frozen** nor **locked** is the destination component.

The destination Data component displays the list of variables defined in the selected module with their values.

Change Format for Variable Value Display

The Debugger allows you to see the value of variables in different formats. This is set by entries in **Format** menu (Table 6.1 on page 198).

Table 6.1 Debugger Display Format

Menu entry	Description
Hex	Variable values are displayed in hexadecimal format.
Oct	Variable values are displayed in octal format.
Dec	Variable values are displayed in signed decimal format.
UDec	Variable values are displayed in unsigned decimal format.
Bin	Variable values are displayed in binary format.
Symbolic	Displayed format depends on variable type.

1. Values for pointer variables are displayed in hexadecimal format.
2. Values for function pointer variables are displayed as function name.
3. Values for character variables are displayed in ASCII character and decimal format.
4. Values for other variables are displayed in signed or unsigned decimal format depending on the variable being signed or not.

Format menu is activated as follows:

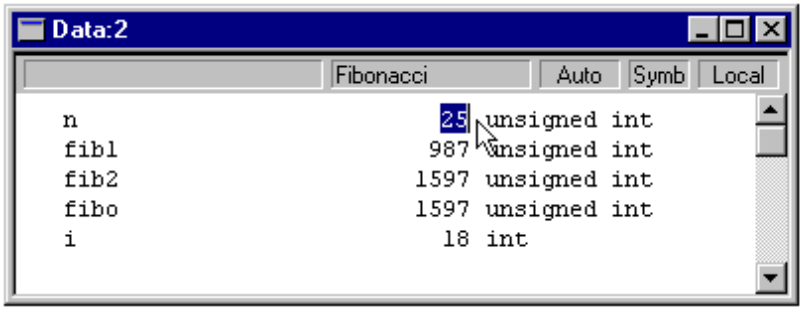
1. Right-click in the Data component. The Data Popup Menu is displayed on the screen.
2. Choose **Format** from Popup Menu. The list of all formats is displayed on the screen.

The format selected is valid for the whole Data component. Values from all variables in the data component are displayed according to the selected format.

Modify a Variable Value

The Debugger allows you to change the value of a variable, as shown in Figure 6.4 on page 199.

Figure 6.4 Modifying a Variable Value



The Debugger allows you to change the value of a variable.

Double-click on a variable. The current variable value is highlighted and can be edited.

1. Formats for the input value follow the rule from ANSI C constant values (prefixed by 0x for hexadecimal value, prefixed by 0 for octal values, otherwise considered as decimal value). For example, if the data component is in decimal format and if a variable input value is 0x20, the variable is initialized with 32. If a variable input value is 020, the variable is initialized with 16.
2. To validate the input value you can either press the **Enter** or **Tab** key.
3. If an input value has been validated by the **Tab** key, the value of the next variable in the component is automatically highlighted (this value can also be edited).
4. To restore the previous variable value, press the **Esc** key or select another variable.

A local variable can be modified when the application is stopped. Since these variables are located on the stack, they do not exist as long as the function where they are defined is not active.

Get the Address Where a Variable is Allocated

The Debugger provides you with the start address and size of a variable if you do the following:

1. Point to a variable name in a Data Component
2. Click the variable name

The start address and size of the selected variable is displayed in the Data info bar.

Inspect Memory Starting at a Variable Location Address

The Debugger provides two ways to dump the memory starting at a variable allocation address.

- Using Drag and Drop

1. Drag a variable name from the Data Component to Memory component.

- Holding down the left mouse button and pressing the **A** key

1. Point to a variable name in a Data Component.
2. Hold the left mouse button down and press the **A** key.

The memory component scrolls until it reaches the address where the selected variable is allocated. The memory range corresponding to the selected variable is highlighted in the memory component.

Load an Address Register with the Address of a Variable

The Debugger allows you to load a register with the address where a variable is allocated.

1. Drag a variable name from the Data Component to Register component.

The destination register is updated with the start address of the selected variable.

How To Work on the Register

This section describes how to work with the Register component.

Change Format of Register Display

The Debugger allows you to display the register content in hexadecimal or binary format.

1. Right-click in the Register component. The Register Popup Menu is displayed on the screen.
2. Choose Options .. from the Popup Menu. The pull down menu containing the possible formats is displayed.
3. Select either binary or hexadecimal format.

The format selected is valid for the Register component. The contents from all registers are displayed according to the selected format.

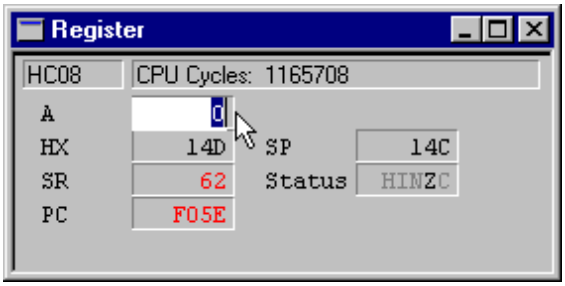
Modify a Register Content

The Debugger allows you to change the content of indexes, accumulators or bit registers.

Modify Index or Accumulator Register Content

Double-click a register. The current register content is highlighted and may be edited.

Figure 6.5 Modifying Index or Accumulator Register Content



1. The format of the input value depends on the format selected for the data component. If the format of the component is **Hex**, the input value is treated as a Hex value. If the input value is 10 the variable will be set to 0x10 = 16.
2. To validate the input value you can either press the **Enter** or **Tab** key, or select another register.

3. If an input value has been validated by the **Tab** key, the content of the next register in the component is automatically highlighted. This register can also be edited).
4. To restore the previous register content, press the **Esc** key.

Modify Bit Register Content

In a bit register, each bit has a specific meaning (a Status Register (SR) or Condition Code Register (CCR).

Mnemonic characters for bits that are set to 1 are displayed in black, whereas mnemonic characters for bits that are reset to 0 are displayed in grey.

Single bits inside the bit register can be toggled by double-clicking the corresponding mnemonic character.

Start Memory Dump at Address Where Register Is Pointing

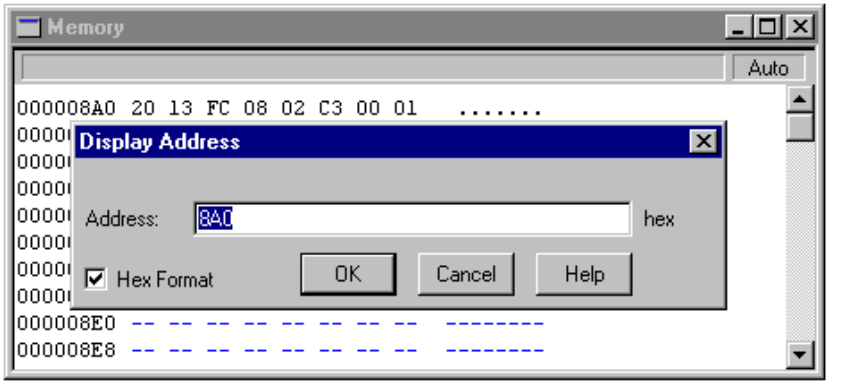
The Debugger provides two ways to dump memory starting at the address a register is pointing to.

- Using Drag and Drop

1. Drag a register from the Register component to Memory component.

- Choose **Address ..**

Figure 6.6 Memory menu Display Address



1. Right-click in the Memory component. The Memory Popup Menu is displayed.
2. Choose **Address** ... from the Popup Menu. The **Memory** ... dialog box shown in Figure 6.6 on page 202 is opened.
3. Enter the register content in the Edit Box and choose **OK** to close the dialog box.

The memory component scrolls until it reaches the address stored in the register.

This feature allows you to display a memory dump from the application stack.

NOTE If “Hex Format” is checked, numbers and letters are considered to be hexadecimal numbers. Otherwise, expressions can be typed and Hex numbers should be prefixed with “Ox” or “$”.

Modify Content of Memory Address

The Debugger allows you to change the content of a memory address. Double-click the memory address you want to modify. Content from the current memory location is highlighted and can be edited.

1. The format for the input value depends on the format selected for the Memory component. If the format for the component is **Hex**, the input value is treated as a Hex value. If input value is 10 the memory address will be set to 0x10 = 16.
2. Once a value has been allocated to a memory word, it is validated and the next memory address is automatically selected and can be edited.
3. To stop editing and validate the last input value, you can either press the **Enter** or **Tab** key, or select another variable.
4. To stop editing and restore the previous memory value, press the **Esc** key.

How to Consult Assembler Instructions Generated by a Source Statement

The Debugger provides an on-line disassembly facility, which allows you to disassemble the hexadecimal code directly from the Debugger code area. Online disassembly can be performed in one of the following ways:

Using Drag and Drop

1. In the Source component, select the section you want to disassemble.
2. Drag the highlighted block to the Assembly component.

Holding down the left mouse button and pressing the R key

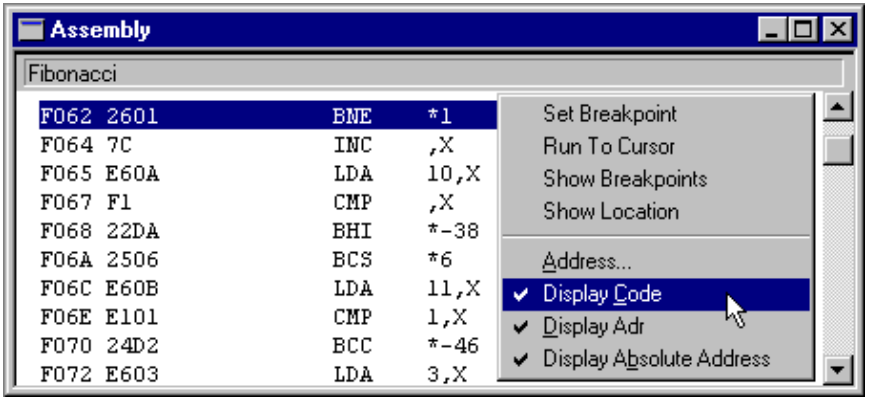
1. In the Source component window, point to the instruction you want to disassemble.
2. Hold down the left mouse button and press the R key

The disassembled code associated with the selected source instruction is greyed in the Assembly component.

How To View Code

The Debugger allows you to view the code associated with each assembler instruction.

Figure 6.7 Viewing Code Associated with Assembler instruction.



Online disassembly can be performed in one of the following ways:

- Using Popup Menu

1. Point in the Assembly component and right-click. The Assembly Popup Menu is displayed.
2. Choose **Display Code** (Figure 6.7 on page 204).

- Using Assembly Menu

1. Click the title bar of the Assembly component. The Assembly menu appears in the debugger menu bar.
2. Choose **Assembly > Display Code**

The Assembly component displays the corresponding code on the left of each assembler instruction.

How to Communicate with the Application

The Debugger has a pseudo-terminal facility. Use the **TestTerm or Terminal** component window to communicate with the application using specific functions defined in the `TERMINAL.H` file and used in the calculator demo file.

1. Start the Debugger and choose **Open...** from the Component menu.
2. Open the **TestTerm or Terminal** Component.
3. Choose **Load**... from the Simulator menu.
4. Load the program CALC.ABS.

Data entered in the **TestTerm** or **Terminal** component window through the keyboard will be fetched by the target application with the '**Read**' function. The target application can send data to the Terminal component window of the host with the '**Write**' function.

About startup.cmd, reset.cmd, preload.cmd, postload.cmd

The command files `startup.cmd, reset.cmd, preload.cmd,` and `postload.cmd` are Debugger system command files. All these command files do not exist automatically. They could be installed when installing a new connection.

However, the Debugger is able to recognize these command files and execute them.

- `startup.cmd` is executed when a connection is loaded (the target defined in the **project.ini** file or loaded when you select **Component>Set Connection**).
- `reset.cmd` is executed when you select **"Connection Name" >Reset** in the menu (**Connection Name** is the real name of the connection, such as **MMDS0508**, **SDI**, etc.).
- `preload.cmd` is executed before loading a `.ABS` application file or Srecords file (when you select **"Connection Name">Load...** in the menu).
- `postload.cmd` is executed after loading a `.ABS` application file or Srecords file (when you select **"Connection Name">Load...** in the menu).

Depending on the connection used, other command files can be recognized by the Debugger. Refer to the appropriate connection manual for information and properties of these command files.

7

CodeWarrior Integration

This chapter provides information on how to use and configure the Simulator/Debugger within CodeWarrior using the following software:.

- CodeWarrior IDE - CW08 version 3.1 or later

Click the following link to jump to the corresponding section of this chapter:

- Debugger Configuration on page 207

Debugger Configuration

To configure the Real Time Debugger and True Time Simulator, in the CodeWarrior IDE open the **Target Settings Panel** by clicking on the Targets panel of the IDE main window, then double clicking on the name of your target in the list displayed in this panel. Then, select **Build Extras** as shown in (Figure 7.1 on page 208).

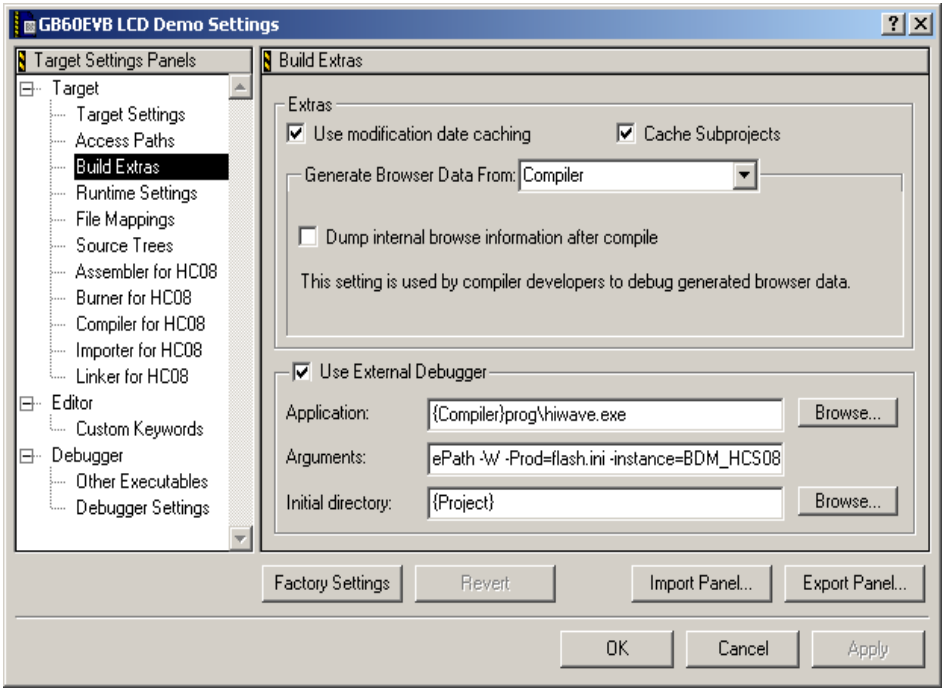
In the **Build Extras** pane check the **Use External Debugger** checkbox.

In the Application field, type the Debugger path, (or select from the Open window by clicking the Browse button)for example: **{Compiler}prog\hiwave.exe** .

In the Arguments field, type the arguments, for example, **%targetFilePath -Target=sim** in the Argument field.

Click on **Apply** to validate these changes.

Figure 7.1 IDE Target Window - Build Extras Panel



8

Debugger DDE Capabilities

Introduction

The DDE is a form of interprocess communication that uses shared memory to exchange data between applications. Applications can use DDE for one-time data transfers and for ongoing exchanges in applications that send updates to one another as new data becomes available.

NOTE **The DDE capabilities of the Debugger are deprecated.** Future versions of the Debugger will have no DDE capabilities. You can use the Component Object Model (COM) Interface.

DDE Implementation

The Debugger integrates a DDE server and DDE client implementation in the KERNEL. The DDE application name of the IDF server is "HI-WAVE".

The Debugger DDE support allows you to execute almost any command that would be available from within the debugger (from Command line). There are also special DDE items for more commonly performed tasks.

This section describes topics and DDE items available to CodeWright clients. In addition to the required System topic, CurrentBuffer and the names of all CodeWright non-system buffers (documents) are available as topics.

Driving Debugger through DDE

The DDE implementation in the Debugger allows you to drive it easily by using the DDE command. To do this, you have to use a program that can send a DDE message (a DDE client application) like DDECLient.exe from Microsoft.

The service name of the Debugger DDE Server is "**HI-WAVE**" and the Topic name for the Debugger DDE Server is "**Command**".

The following example is done with DDECLient.exe from Microsoft.

1. Run the Debugger and in the "Service" field in the DDEClient type: "HI-WAVE"
2. In the "Topic" field type "Command"

3. Push the "Connect" button of the DDEClient. The following message will appear in DDECLient: "Connected to HI-WAVE|Command".
4. In the "Exec" field of DDECLient type a Debugger command, for example "open recorder" and click the "Exec" button. The command is executed by way of DDE and you'll see a new recorder component in the Debugger.

NOTE You can disconnect the DDE in the Debugger. The Debugger can be started without DDE (this is saved in the project file). To view the current state, open a command line component and type the following command: "DDEPROTOCOL STATUS". The state must be: "DDEPROTOCOL ON" to ensure the DDE works properly.

9

Synchronized Debugging Through DA-C IDE

This chapter provides information on how to use and configure Freescale tools within the Development Assistant for C (DA-C) IDE. For more information on DA-C, refer to the "Development Assistant for C" documentation v 3.5.

You must be running:

DA-C - version 3.5 build 555 or later - (Development Assistant for C - RistanCASE).

Click any of the following links to jump to the corresponding section of this chapter:

- Configuring DA-C IDE for Freescale Tool Kit on page 211
- Debugger Interface on page 222
- Synchronized Debugging on page 227
- Troubleshooting on page 227

Configuring DA-C IDE for Freescale Tool Kit

Install the DA-C software. The Freescale CD contains a demo version located in `\Addons\DA-C`. Run **Setup** to install the **Typical** installation.

A few configurations are required in order to make efficient use of Freescale Tools within DA-C IDE.

- Create a new project
- Configure the working directories
- Configure the file types
- Configure the Freescale library path
- Adding files to project
- Building the Database
- Configure the tools

In the following sections, we assume that the Freescale tool kit is installed in "C:\Freescale" directory. You may have to adapt the paths to your current installation. An example configuration for the M68k CPU is provided, which can be adapted to each CPU supported by Freescale.

Create New Project

Start DA-C.exe and choose **Project>New Project…** from the main menu. Browse to the directory and enter a project file name, for example:

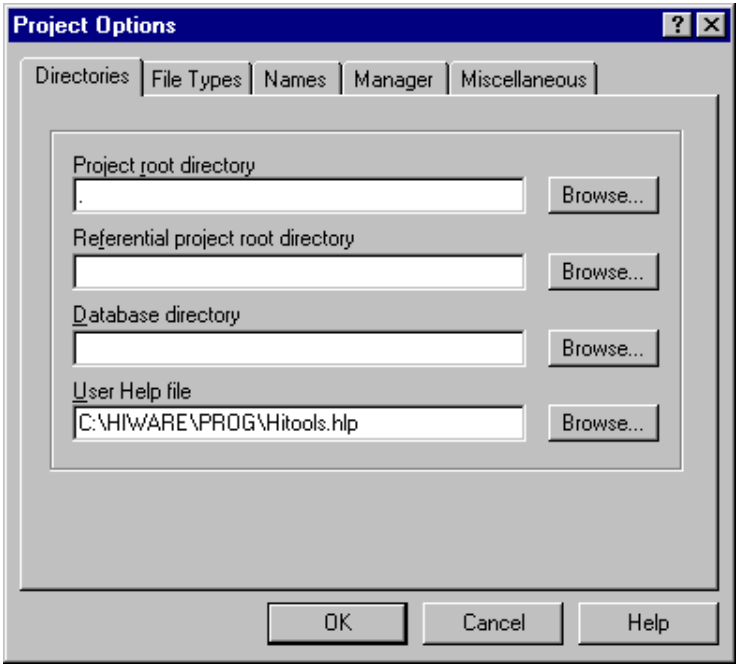
```
"C:\Freescale\work\<processor>c\myproject"
```

Change the <processor> field to your CPU). A specific project file is created with "`.dcp`" extension (for example "`myproject.dcp`").

Configure Working Directories

Choose **Options>Project** from the main menu of DA-C. The window shown in Figure 9.1 on page 212 contains options, which establish directories for the project.

Figure 9.1 DA-C Project Options Window - Directories Tab



Project Root Directory

This text box determines the project root directory. The full path is expected, or a single dot can be entered, which stands for the same directory where the project file resides. All files that belong to the project are considered relative to the Project root directory, if the full path of the file is not given. In our case, keep the single dot for the project root directory.

Referential Project Root Directory

If not empty, this text box specifies an alternate Project Root Path for searching files not found in the original project path. Filenames in the original path with referential extensions are tried before those in the referential path. Specified path may be either full or relative to project root, and it may not specify a subdirectory in the project root directory tree. Leave this field empty.

Database Directory

This text box determines the directory where the symbols and software metrics database will be saved. This directory can be absolute or relative to the Project Root Directory. Leave this field empty.

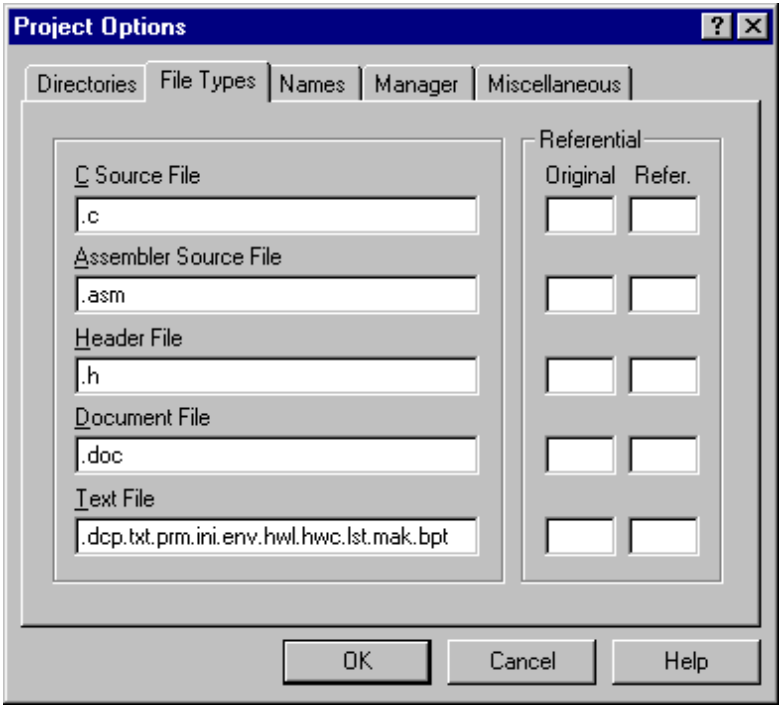
User Help File

This text box determines the user help file, for example compiler help file. The hot key for User Help File can be defined in the Keyboard definition file (default Ctrl-Shift-F1). Browse in the "\prog" directory of your Freescale installation and select the help file matching your CPU.

Configure File Types

From the main menu of DA-C choose "File Types" to configure the basic file types. The File Types Tab of the Project Options Window contains options, which determine file types of the project. For an efficient use of Freescale tools, Figure 9.2 on page 214 shows file extension types that can be defined.

Figure 9.2 DA-C Project Options Window - File Types Tab

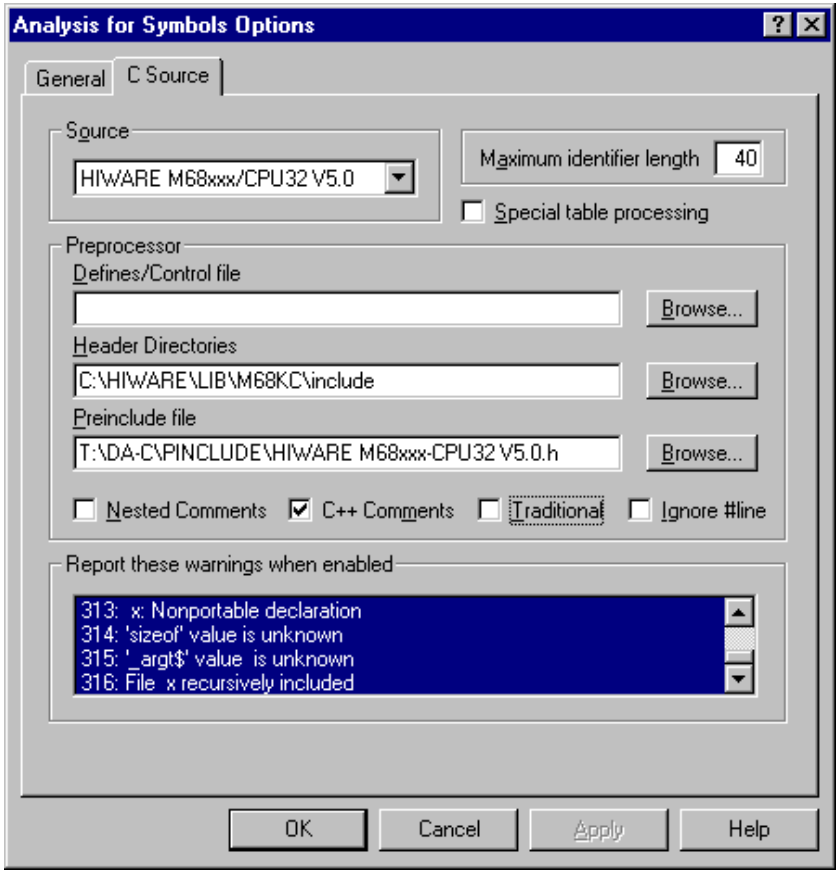


Configure Library Path

An additional configuration path must be defined to specify the location of library header files (needed for DA-C symbol analysis). This can be done by choosing **Options>Analysis for Symbols … >C Source** in the main menu of DA-C.

The window shown in Figure 9.3 on page 215 contains options that determine parameters of the C source code analysis.

Figure 9.3 Analysis for Symbols Options Window - C Source Tab



Source

The supported C dialects of the C language used in the current project can be selected in this text field. In our example we chose the Freescale M68k language (adapt it to your needs).

Preprocessor - Header Directories

This text box determines the list of directories that are to be searched for files named within the "#include" directive. A semicolon separates directories. Only listed directories are searched for files, named between "<" and ">". Searching for files, named between quotation marks (""), starts in the directory of the source file containing "#include" directive.

The list of header directories can be assigned in a file. In that case, this field contains the file name (absolute or relative in relation to the project root) with prefix @. Directories are separated with a semi-colon or new line.

Define the library path matching your CPU (assuming Freescale tools are installed on "`C:\Freescale`"):

```
C:\Freescale\lib\<processor>c\include.
```

Preprocessor - Preinclude File

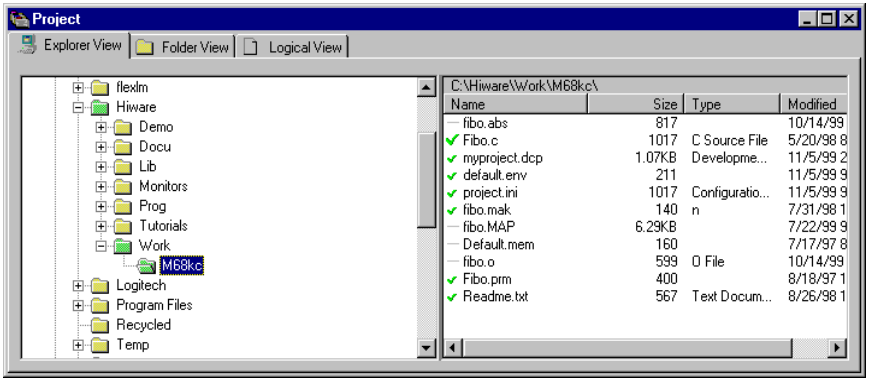
This text box determines the name of the file that will be included automatically at the beginning of every source module during analysis, in the same way as if #include "string" were present in the first line. The preinclude file can be used to specify predefined macros and variable and function declarations for a particular compiler, which are not set by default in DA-C analysis. We have selected the one corresponding to our example: M68k preinclude file (adapt it to your needs).

Adding Files to Project

In the Project Window the Explorer View Tab replaces the Window's Explorer and supplies you with additional information on directories containing project files. It also gives you the option to add files into the project. For example, we will now set all files needed to run the "fibo" example.

In the Explorer View, browse to the ">Freescale>WORK><processor>c" directory of your Freescale installation and select "`fibo.c`" file. Then right-click mouse button and choose "**Add to Project**". The file is now added in the current project and a green mark appears in front of it (Figure 9.4 on page 216).

Figure 9.4 Adding Files to Project Using Explorer Tab



In the same way, select "`fibo.prm`" file and add it to this project.

You can also add a directory to the project in the following way:

- Select Explorer View Tab in Project Window.
- In the left section, select the directory with files to be added to the project (files from subdirectories may also be added to the project).
- From popup menu choose "**Add to project**".

This operation may also be performed from Folder view, if the directory is in the left section.

NOTE When adding an entire directory to the project only files with extensions defined in **Options>Project>File types** (as described in section "Configure the file type") will be added to the project.

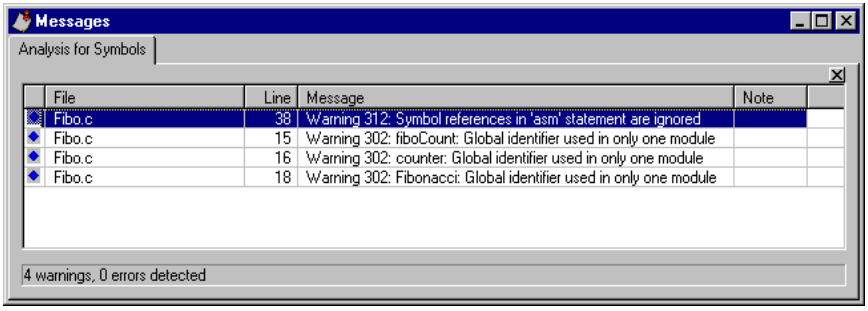
Building The Database

Development Assistant for C provides the static code analysis of C source files, as well as generating various data based on the results.

Analysis of the project source files and generation of the database are divided into two phases: the analysis of individual program modules and generation of data about global symbols usage. Results of the analysis are saved in database files on the disk, which enables their later use in DA-C. You can choose between the unconditional analysis of all project files and the analysis of changed source files only, using **Start> Build database** and **Start>Update database** commands. The latter one will optionally check if the include files used in program modules are changed as well.

To build the database in our example use **Start>Build** database command, which makes the unconditional analysis of all project files and creates a database containing information about analyzed source code. Errors and Warnings detected during this operation are displayed in the Messages window as illustrated in Figure 9.5 on page 217 (for `Fibo.c` sample file):

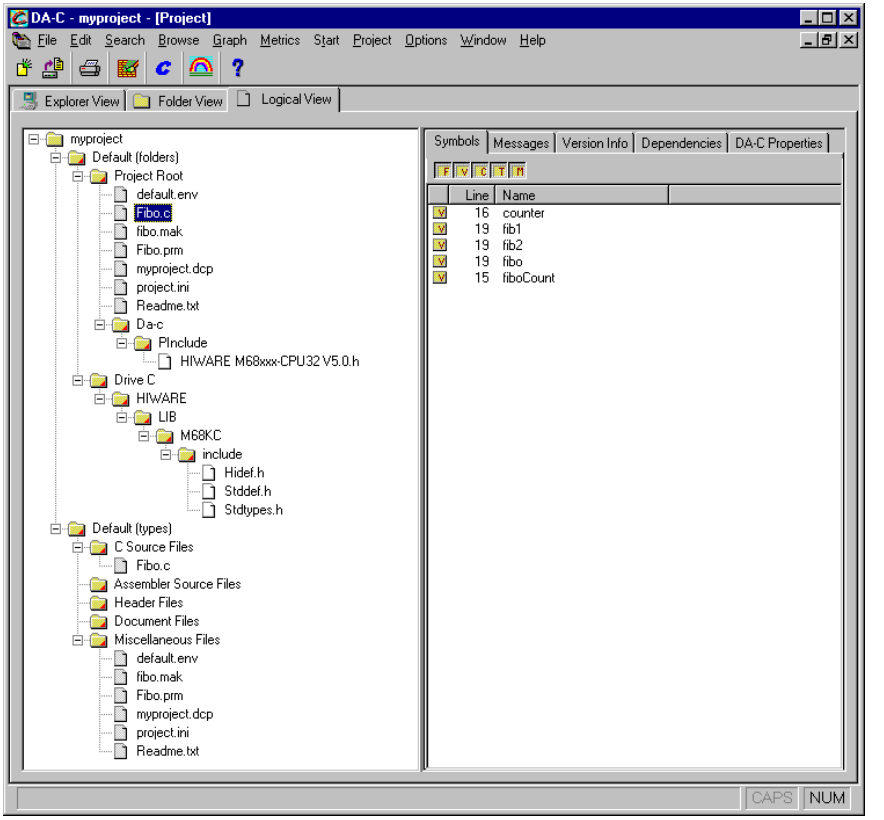
Figure 9.5 DA-C Message Window



After the analysis of all project files, the new database file containing information about global symbols is constructed. Refer to the DA-C manual for more information on how symbol information can be used.

In the Project Manager's window of DA-C, select the **Logical View** Tab shown in Figure 9.6 on page 218 and unfold all fields, you will now have the overview of your project.

Figure 9.6 Logical View Tab



Double-click on "`Fibo.c`" file to open it.

Configuring The Tools

We will now configure the compiler and maker into DA-C IDE. Procedures are defined in **Project>User Defined Actions…** from the main menu of DA-C.

Compiler Configuration

In **Menu "Start" Actions**, click on **new** and fill in the **New Action** box with "C&ompile", then press ENTER (Figure 9.7 on page 219). In the **Toolbar** field, you can associate a bitmap with each tool, for example click on the **Picture** radio button and browse to the "\Bitmap" directory of your current DA-C installation and choose Compiler.bmp. This is a default bitmap delivered with DA-C IDE. Here you are able to add your own bitmap.

Figure 9.7 DA-C Compiler Settings

Now fill in the **Action Script** field in order to associate related compiler actions. Copy the following lines shown in Listing 9.1 on page 220 in the Action Script field and change the directory to where the compiler is located.

Listing 9.1 Script for Compiler Action Association

```
.%If(%HasModuleExt(%CurrFile),,%Message(Not a module file!)%Cancel)
.%SaveAll
.c:\Freescale\prog\cm68k.exe %CurrFile
.%if(%Exist(edout),,%Message(No Messages found!)%Cancel)
.%ErrClr(Compiler)
.%ErrGet(edout,Compiler)
.%Reset(%CurrFile)
```

Click on **OK** to validate these settings. Select "`Fibo.c`" file. Click on the "Compiler" button (or from the main menu of DA-C select **Start>Compile**). This file is now compiled and the corresponding object file ("`Fibo.o`") is generated.

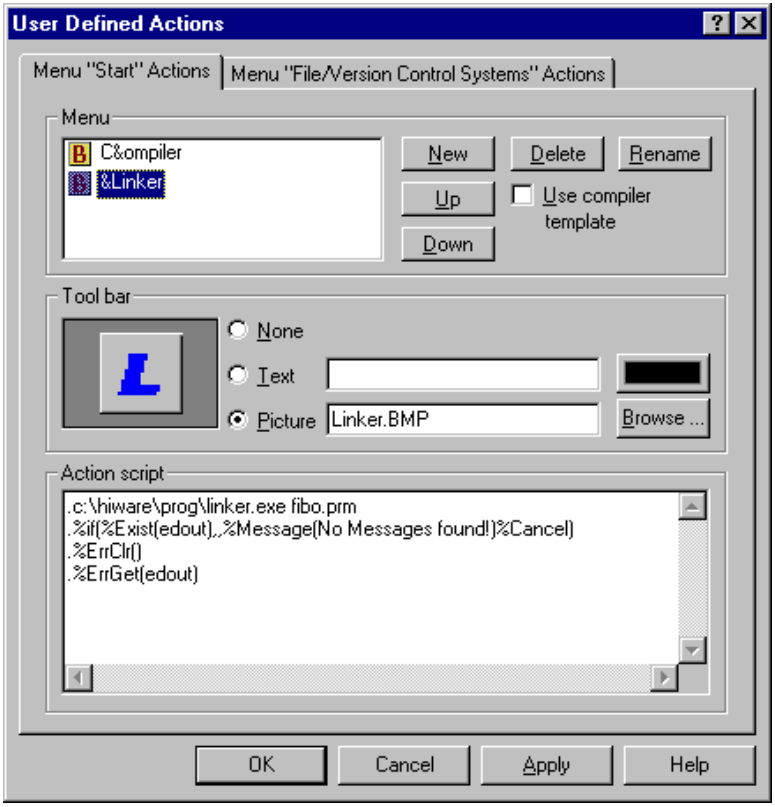
Linker Configuration

In the same way, you can now configure the linker as illustrated in Figure 9.8 on page 221. In the **Menu "Start" Actions**, click on new and fill in the created **New Action** box with " &Link", then validate with ENTER. After setting the corresponding bitmap, copy the following lines shown in Listing 9.2 on page 220 in the **Action Script** field and change the directory to where the linker is located.

Listing 9.2 Script for Linker Action Association

```
+c:\Freescale\prog\linker.exe fibo.prm
.%if(%Exist(edout),,%Message(No Messages found!)%Cancel)
.%ErrClr()
.%ErrGet(edout)
```

Figure 9.8 DA-C Linker Settings



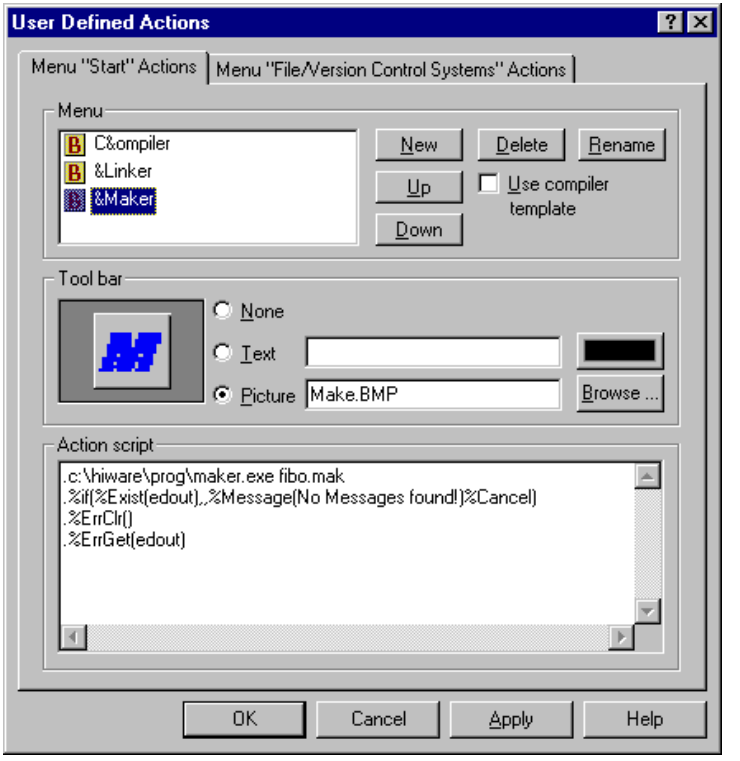
Maker Configuration

In the same way, you can now configure the maker as illustrated in Figure 9.9 on page 222. In the **Menu "Start" Actions,** click on new and fill in the created **New Action** box with " **&Make"**, then press ENTER. After setting the corresponding bitmap, copy the lines from Listing 9.3 on page 221 in the **Action Script** field and change the directory to where the maker is located.

Listing 9.3 Script for Maker Action Association

```
+c:\Freescale\prog\maker.exe fibo.mak
.%if(%Exist(edout),,%Message(No Messages found!)%Cancel)
.%ErrClr()
.%ErrGet(edout)
```

Figure 9.9 DA-C Maker Settings



Debugger Interface

DA-C v3.5 is currently integrating a DAPI interface (Debugging support Application Programming Interface). Through this interface DA-C is enabled to exchange messages with the Debugger. The advantages of such connection show that it is possible to set or delete break points from within DA-C (in an editor, flow chart, graph, browser) and to execute other debugger operations. DA-C is following Debugger in its operation, since it is always in the same file and on the same line as the debugger. Thus, usability of both the DA-C and Debugger is increased. Some configurations are required in order to make an efficient use of this Debugger Interface:

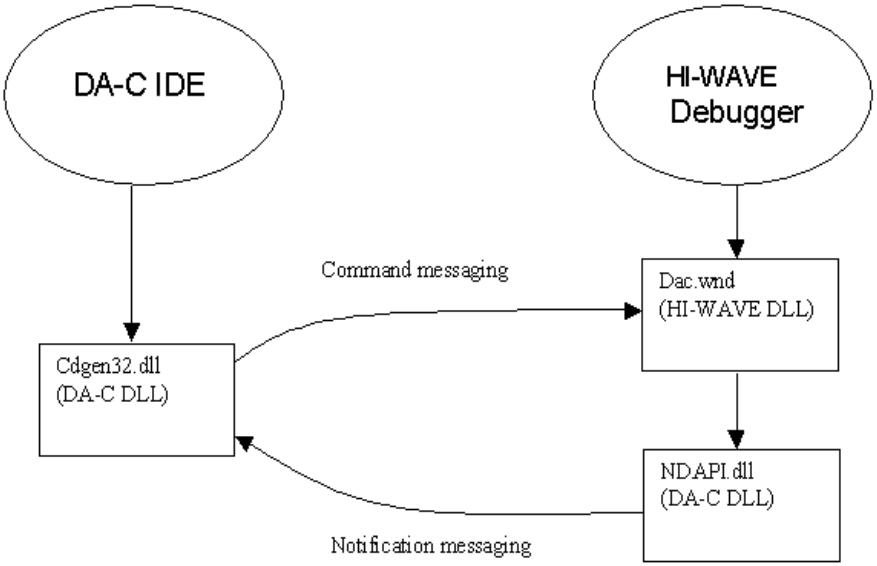
- Installation of communication DLL
- Configuration of Debugger properties
- Configuration of the Debugger project file

DA-C IDE and Debugger Communication

DA-C and the Debugger are both Microsoft Windows applications and communication is based on the DDE protocol, as shown in Figure 9.10 on page 223. The whole system contains:

- DA-C
- Debugger
- cDAPI interface implementation DLL - which is used by DA-C (Cdgen32.dll)
- nDAPI communication DLL (provided by DA-C), which is used by Debugger
- Debugger specific DLL for bridging its interface to debugging environment and DA-C's nDAPI (DAC.wnd)

Figure 9.10 Communication between DA-C IDE and Debugger



Communication DLL Installation

As described previously, the Debugger needs the nDAPI communication DLL (provided by DA-C IDE). This dll (called Ndapi.dll) is automatically installed during the Freescale

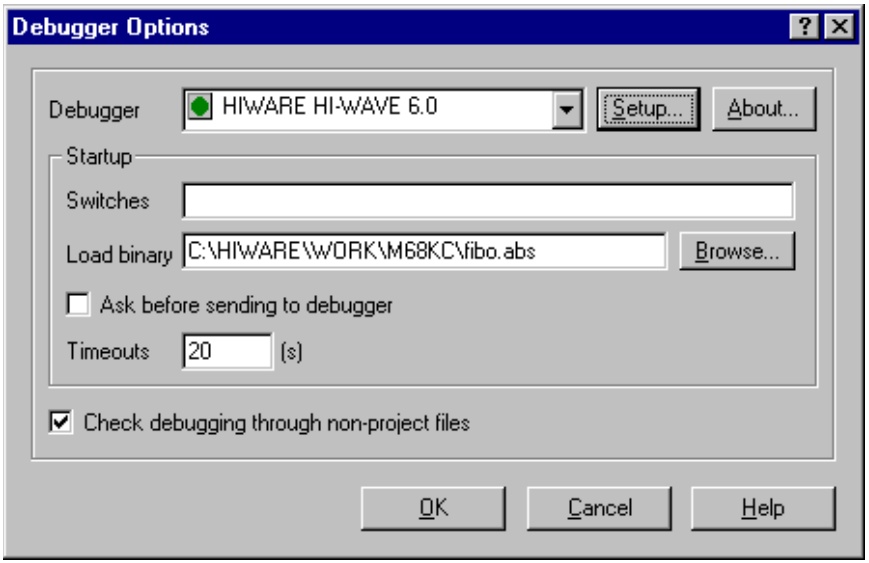
Tool Kit installation. However, if you install a new release of DA-C you have to follow this procedure:

In the "`\Program`" directory of your DA-C installation, copy the "`Ndapi32.dll`" (`Ndapi32.dll` version 1.1 or later) and paste it in your current "`Freescale\PROG`" directory (where Debugger is located). Then rename it to "`Ndapi.dll`".

Debugger Properties Configuration

In the DA-C main menu, choose **Options>Debugger**, the dialog box shown in Figure 9.11 on page 224 is opened.

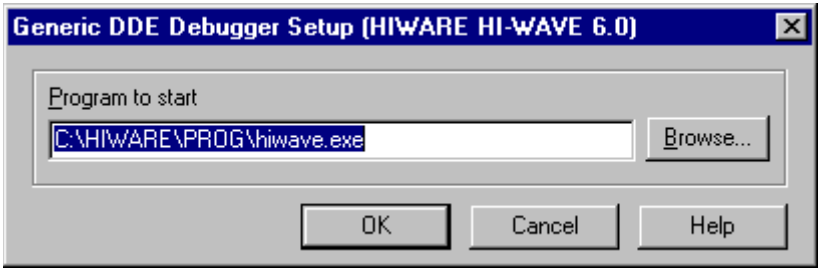
Figure 9.11 DA-C Debugger Options Dialog Box



In the "**Debugger**" combo-box, select the corresponding debugger: "**HI-WAVE 6.0**". Now specify the binary file to be opened: in our example we want to debug the "`fibo.abs`" file.

Then click on the **Setup…** button. The dialog box shown in Figure 9.12 on page 225 is opened.

Figure 9.12 DDE Debugger Setup Dialog Box

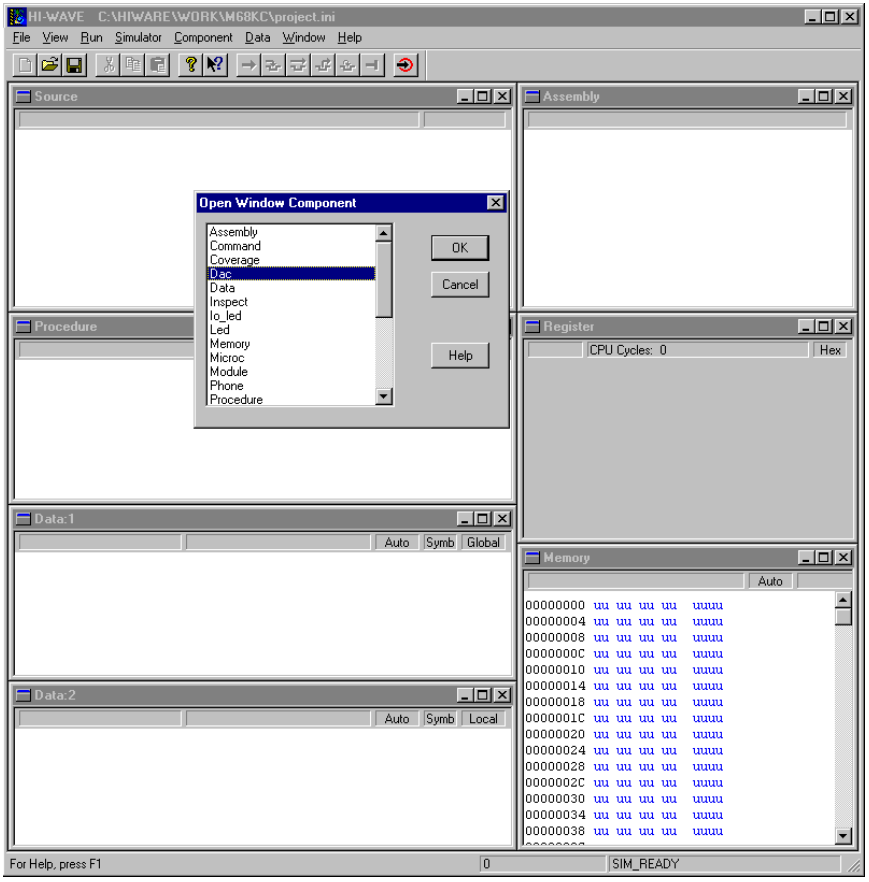


Specify the path to the "**hiwave.exe**" file or use the **Browse**… button then click on **OK**.

Debugger Project File Configuration

Before configuring the project file, close DA-C. Open Debugger (for example, from a shell) and select **File>Open Project**… from the main menu bar. Select the "`Project.ini`" file from the currently defined working directory (in our case "`C:\Freescale\WORK\<processor>c\project.ini`"). We will now add in the layout of the project the Debugger DAC component ("`dac.wnd`"). In the Debugger select **Component >Open** from the main menu bar and choose "**Dac**", as shown in Figure 9.13 on page 226.

Figure 9.13 DA-C Component Opening



The Debugger DAC window, which is needed for communication with DA-C IDE is now opened (Figure 9.14 on page 226).

Figure 9.14 DA-C Window

You have to save this configuration by selecting **File>Save Project** from the main menu of the Debugger. This component will be automatically loaded the next time this project is called. Close the Debugger.

Synchronized Debugging

We can now test the synchronization between DA-C IDE and Debugger. Run **DA-C.exe** and open the project previously created. Open "Fibo.c" if it's not already open. Right-click mouse button on "`Fibo.c`" source window and select "main" in the popup menu. The cursor points to the "void main(void) {" statement. In the main menu from DA-C, select **Debug**>**Set Breakpoint** (or click on the corresponding button on the debug toolbar), the selected line is highlighted in red, indicating that a breakpoint has been set. Then select **Debug**>**Run**, the Debugger is now started and after a while stops on the specified breakpoint. Up to now, you can debug from DA-C IDE with the toolbar, as shown in Figure 9.15 on page 227 or from the Debugger.

Figure 9.15 DA-C toolbar

NOTE In case of changes to your source code, don't forget to rebuild the Database when generating new binary files to avoid misalignment between the Debugger and DA-C source positions.

Troubleshooting

This section describes possible trouble when trying to connect the Debugger with the DA-C IDE.

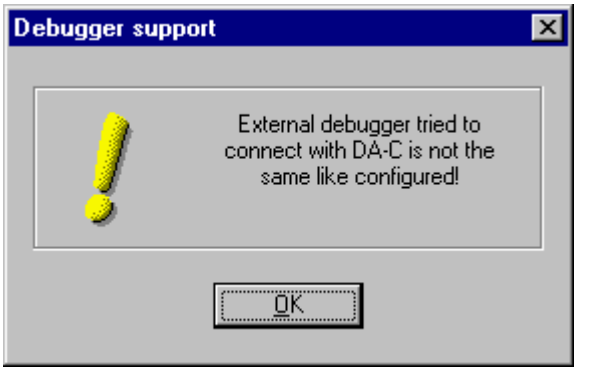
1. When loading DAC component into the Debugger, if the message box shown in Figure 9.16 on page 227 is displayed:

Figure 9.16 DA-C Component Loading Error Message

check if the `Ndapi.dll` is located in the "`\prog`" directory of your current Freescale installation. If not, copy the specified DLL into this directory.

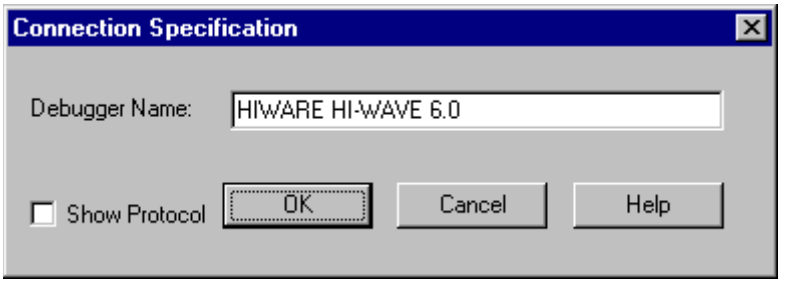
2. If the message box shown in Figure 9.17 on page 228 is displayed in DA-C IDE:

Figure 9.17 DA-C Debugger Support Message



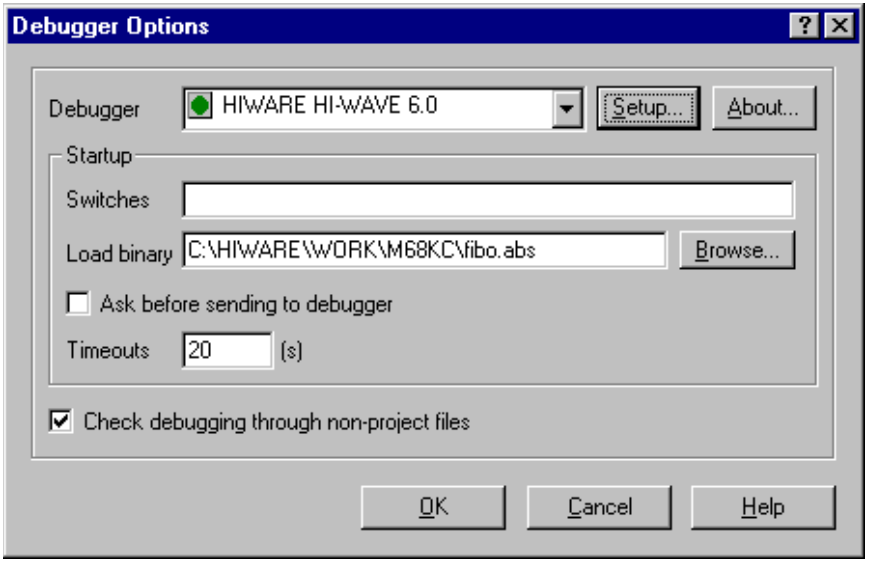
This means that the current name specified in the **Options>Debugger** main menu of DA-C doesn't match the debugger name specified in the Debugger. Open the setup dialog in the Debugger by clicking on the DA-C Link component and choose **DA-C Link>Setup…** from the main menu. The "Connection Specification" dialog box is opened (Figure 9.18 on page 228).

Figure 9.18 DA-C Connection Specification Dialog Box



Compare the "**Debugger Name**" from this dialog box with the selected Debugger in DA-C IDE (**Options>Debugger**), as shown in Figure 9.19 on page 229.

Figure 9.19 DA-C Debugger Options Dialog Box



Both must be the same. If it's not the case, change it in the Debugger "Connection Specification" and click **OK**. This implies a new connection to be established and the "Connection Specification" to be saved in the current "`Project.ini`" file in the section shown in Listing 9.4 on page 229.

Listing 9.4 DA-C Section in Project File.

```
[DA-C]
DEBUGGER_NAME=HI-WAVE 6.0
SHOWPROT=1
```

Book II - HC08 Debug Connections

Book II Contents

Each section of the Debugger manual includes information to help you become more familiar with the Debugger, to use all its functions and help you understand how to use the environment. This book, the HC08 Debugger Connections, defines the connections available for debugging code written for HC08 CPUs.

This book consists of the following sections:

- "HC08/RS08 Debugging First Steps" on page 233
- "HC08 Full Chip Simulation" on page 249
- "MON08 Interface Connection" on page 329
- "ICS MON08 Interface Connection" on page 345
- "P&E Multilink/Cyclone PRO Connection" on page 361
- "ICS P&E Multilink/Cyclone PRO Connections" on page 381
- Softec HC08 Connection on page 395
- FSICEBASE Emulator on page 403
- MMDS0508 Emulator on page 431
- MMEVS0508 Emulator on page 481

10

HC08/RS08 Debugging First Steps

Since the initial steps for creating a project are similar, the following information provides an example of the first few steps of the New Project wizard that apply to creating an HC(S)08, HC08 or RS08 project. Debugging code using the CodeWarrior IDE requires that a project be created or exists, which specifies a connection that can be used to debug the code. This section guides you through the first steps toward code debugging with CodeWarrior and the following connections:

- Full Chip Simulation connection
- HC08 Serial Monitor connection
- ICS MON08 Interface connection
- P&E Multilink/Cyclone Pro connection
- ICS P&E Multilink/Cyclone Pro connection
- SofTec HC08 connection

NOTE The initial project creation steps for RS08 Connections mentioned in "Book IV - RS08 Debugger Connections" on page 605 are similar to the first steps mentioned in "Debugging First Steps Using the Wizard" on page 236.

Technical Considerations

While they can be used to debug code, some of these connections have special technical considerations, as discussed in the following paragraphs.

Full Chip Simulation Considerations

The Full Chip Simulation (FCS) connection runs a complete simulation of all processor peripherals and I/O on the user's PC. No development board is required. Each derivative has a totally different simulation engine to accurately simulate the memory ranges, I/O, and peripherals for any given derivative.

HC08 Serial Monitor Considerations

The 8/16 bit debugger (and then the CodeWarrior IDE) might be connected to HC08 hardware using the HC08 Serial Monitor connection. This connection supports communication specifications described in the application note **f**rom Freescale.

When the debugger runs the serial monitor connection, it can communicate and debug hardware running the serial monitor in full compliance with the Freescale Application Note specifications. Please refer to this Application Note for communication hardware requirements.

ICS MON08 Interface Connection

In-Circuit Simulation (ICS) Mode is a P&E Microcomputer Systems mode of operation that is a hybrid between In-Circuit Debugging, and Full Chip simulation. In-Circuit Simulation, or ICS, mode simulates the CPU core instructions on the user's PC.

P&E Multilink/Cyclone Pro Considerations

In order to use the **P&E Cable 12** or **P&E BDN-Multilink**, the drivers from P&E must be installed on the host computer.

A parallel cable should be used for communication between the **P&E Cable 12** or **BDM-Multilink** and the host computer.

The communication protocol between the **P&E cable 12** or **BDM-Multilink** and the host is handled by the *unit_12z.dll* target driver which is automatically loaded with the connection.

ICS P&E Multilink/Cyclone Pro Considerations

In-Circuit Simulation Mode is a P&E Microcomputer Systems mode of operation that is a hybrid between In-Circuit Debugging, and Full Chip simulation.

SofTec HC08 Considerations

The 8/16 bits debugger (and then the CodeWarrior IDE) might be connected to HC08 hardware using the SofTec HC08.

When the debugger runs the **SofTec HC08** connection, it can communicate and debug **CPU08 (HC08)** core based hardware connected throught the SofTec in-circuit debugger/ programmer units, i.e:

SofTec Microsystems HC08 ISP Debuggers/Programmers (inDART Series) and Starter Kits (AK/SK/PK/ZK and newer Series).

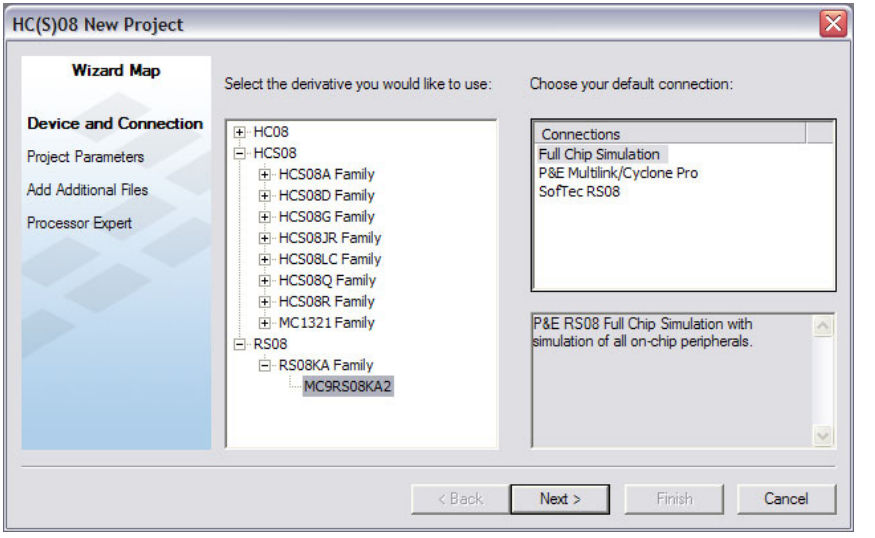
Please refer to the "***inDART®-HC08 In-Circuit Debugger/Programmer for Motorola HC08 Family FLASH Devices User's Manual***" from SofTec for communication hardware requirements and SofTec product installation.

Debugging First Steps Using the Wizard

To take the first steps toward debugging with CodeWarrior IDE using the stationery Wizard:

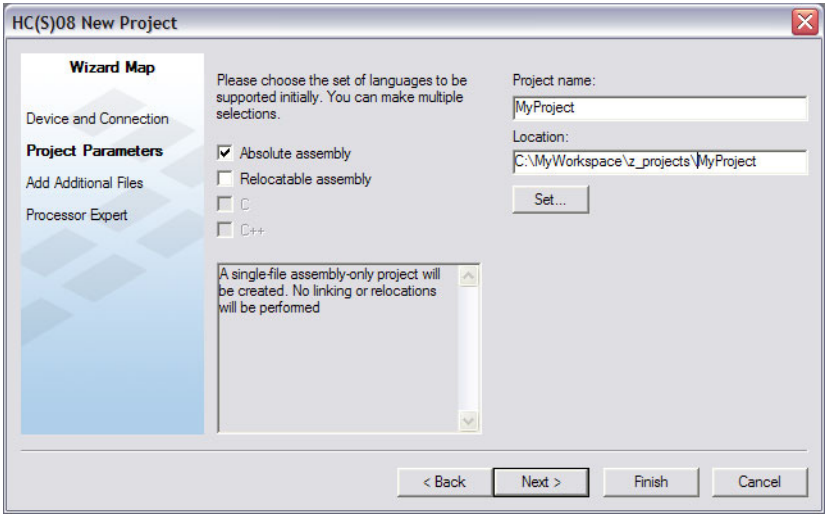
1. Run the *CodeWarrior IDE* with the shortcut created in the program group.
2. Choose the menu **File > New Project...** to create a new project from stationery - the ***HC(S)08 New Project*** Wizard first screen appears.

Figure 10.1 New Window - Project Tab



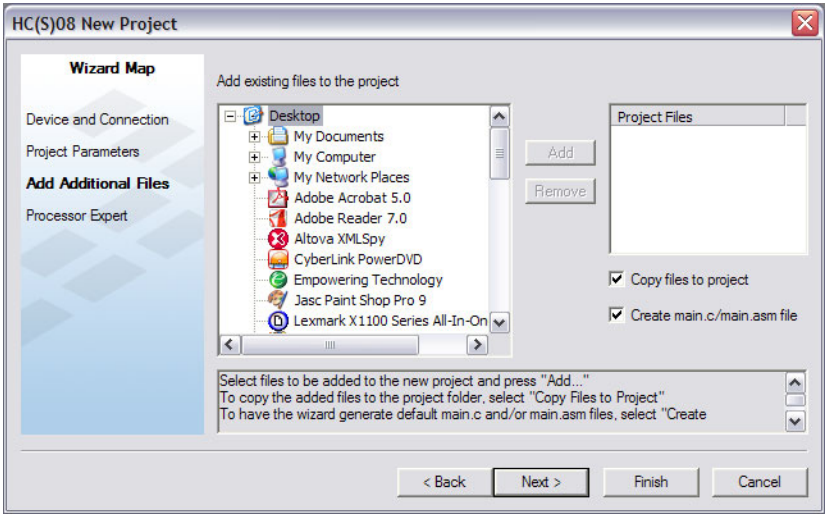
3. Expand the list and select the cpu derivative for your new project
4. After selecting derivative, choose connection from list that appears.
5. Click the **Next** button to proceed.

Figure 10.2 Project Name Screen



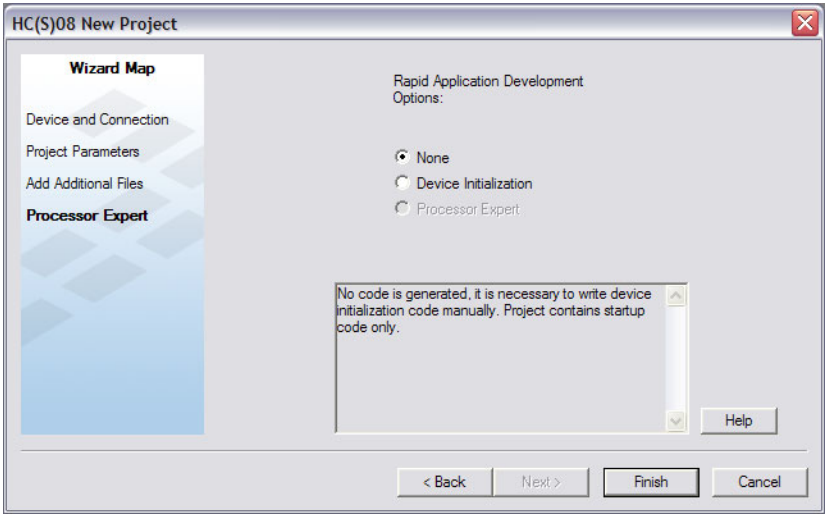
6. Choose the languages to be supported.
   - Assembly - If only Assembly is selected, you can later choose to use absolute/single file assembly application or relocatable assembly.
   - C - This will set up your application with an ANSI C-compliant startup code, doing initialization of global variables.
   - C++ - This will set up your application with an ANSI C++ startup code, doing global class object initialization.
7. Enter a project name and folder location to store project.
8. Click the **Next** button to proceed.

Figure 10.3 Add Files to Project Screen



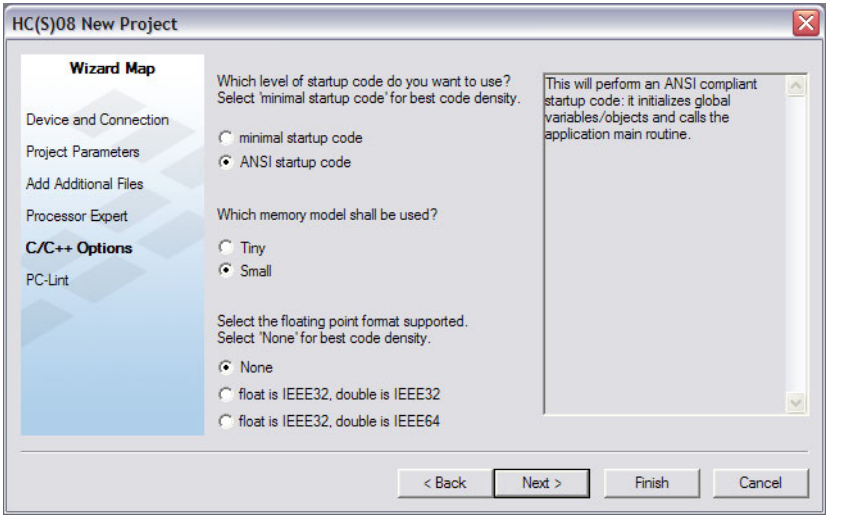
9. If needed, browse to and add existing files to project.
10. Click the **Next** button to proceed.

Figure 10.4 Processor Expert Screen



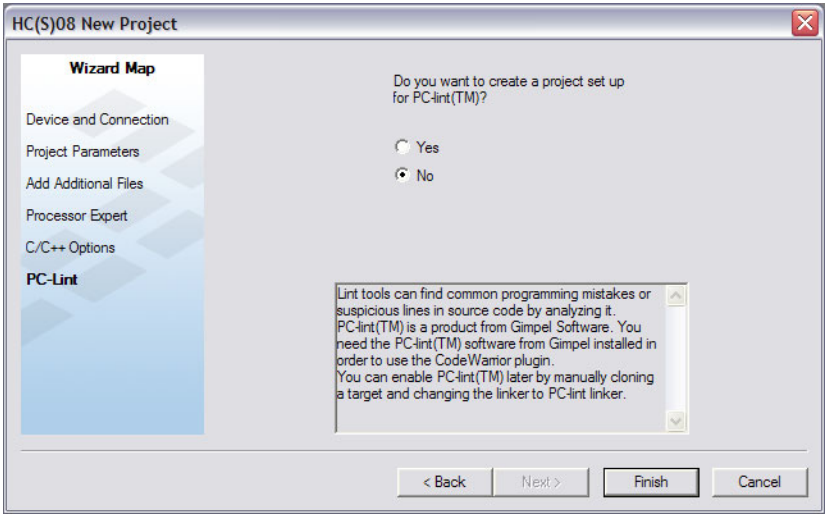
11. Select a Rapid Application Development option.
12. Depending on selected cpu derivative this may be last the screen, click on the **Finish** button - the IDE opens.

Figure 10.5 C/C++ Options Screen


HC(S)08 New Project
Wizard Map
Device and Connection
Project Parameters
Add Additional Files
Processor Expert
C/C++ Options
PC-Lint
Which level of startup code do you want to use?
Select 'minimal startup code' for best code density.
minimal startup code
ANSI startup code
Which memory model shall be used?
Tiny
Small
Select the floating point format supported.
Select 'None' for best code density.
None
float is IEEE32, double is IEEE32
float is IEEE32, double is IEEE64
This will perform an ANSI compliant startup code: it initializes global variables/objects and calls the application main routine.
< Back
Next >
Finish
Cancel

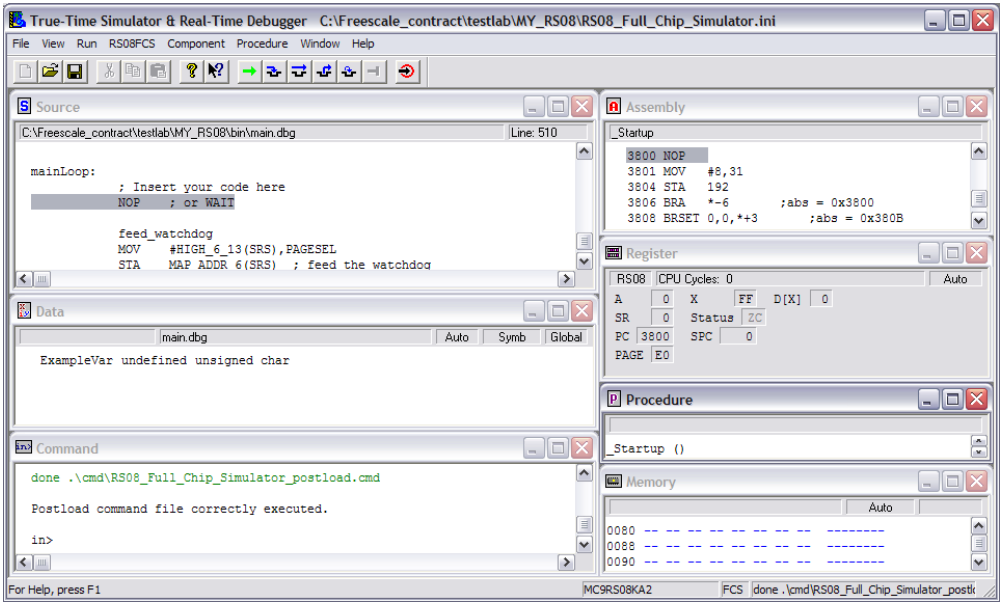
13. Depending on cpu derivative selected, additional screens may appear; select startup code, memory model, and floating point format.
    - Minimal startup code - This startup code initializes the stack pointer and calls the main function. No initialization of global variables is done, giving the user the best speed/code density and a fast startup time. But, the application code has to care about variable initialization. This makes this option not ANSI compliant, since ANSI requires variable initialization.
    - ANSI startup code - This performs an ANSI-compliant startup code that initializes global variables/objects and calls the applicaiton main routine.
    - Small - The Small memory model is best used if both the code and the data fit into the 64kB address space. By default all variables and functions are accessed with 16-bit addresses. the compiler does support banked functions or paged variables in this memory model, but all accesses have to be explicitly handled.
    - None - Don't use floating point for the HC08.
    - Float is IEEE32, double is IEEE32 - All float and double variables are 32-bit IEEE32 for the HC08.
    - Float is IEEE32, double is IEEE64 - Float variables are 32-bit IEEE32. Double variables are 64-bit IEEE64 for the HC08.
14. Click **Next** button to proceed

Figure 10.6 PC-Lint screen


HC(S)08 New Project
Wizard Map
Device and Connection
Project Parameters
Add Additional Files
Processor Expert
C/C++ Options
PC-Lint
Do you want to create a project set up for PC-lint(TM)?
Yes
No
Lint tools can find common programming mistakes or suspicious lines in source code by analyzing it. PC-lint(TM) is a product from Gimpel Software. You need the PC-lint(TM) software from Gimpel installed in order to use the CodeWarrior plugin. You can enable PC-lint(TM) later by manually cloning a target and changing the linker to PC-lint linker.
< Back
Next >
Finish
Cancel

15. Depending on cpu derivative selected, choose whether or not to use PC-lint.
16. Click the **Finish** button.
17. In the IDE main window toolbar Project menu, choose **Project > Make.**
18. Now choose **Project > Debug** to start the debugger.

Figure 10.7 Your Project in Debugger Main Window



Switching Connections

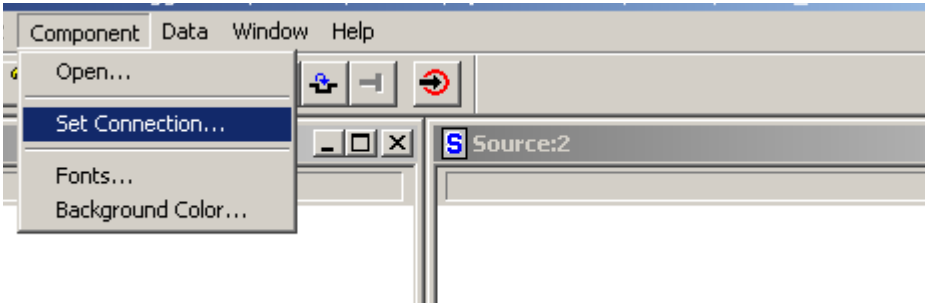
It is possible to switch connections from within an existing debugging project. To switch connections, select **Set Connection...** from the **Component** menu. The following information provides two examples of switching a connection.

Loading the Full Chip Simulation Connection

Because there is no actual hardware involved in switching from another project, such as the SofTec in-Dart HCS08 connection, to the FCS connection, the process is simple. To load the FCS connection from within an existing project, take the following steps:

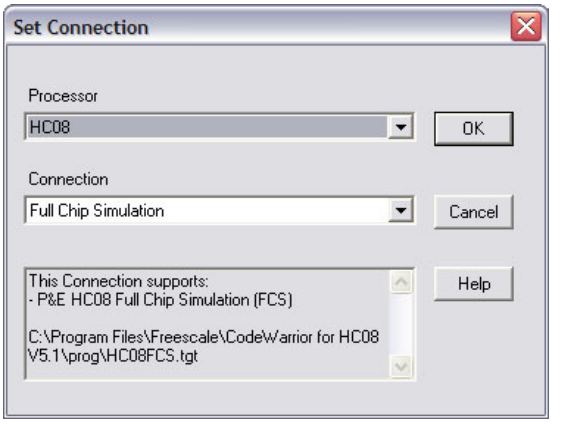
1. From the Debugger main menu, select *Component* | *Set Connection...*, as shown below.

Figure 10.8 Component Menu



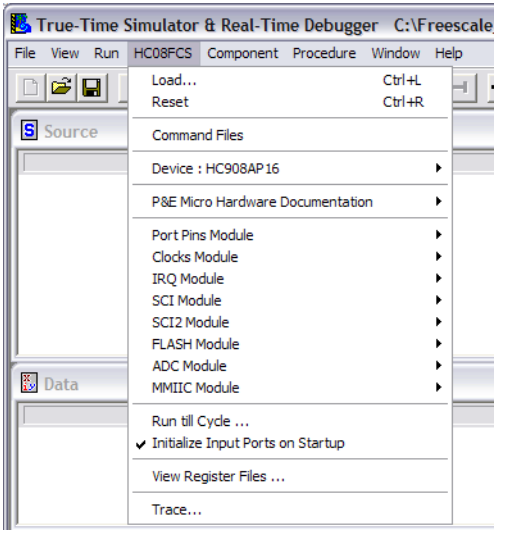
The Set Connection dialog box now appears.

Figure 10.9 Set Connection Dialog Box



2. Select the Processor, for example, HC08 and the Connection as Full Chip Simulation.
3. Press the OK button. The Debugger main menu entry bar for the connection now changes to HC08FCS.

Figure 10.10 HC08 FCS Menu



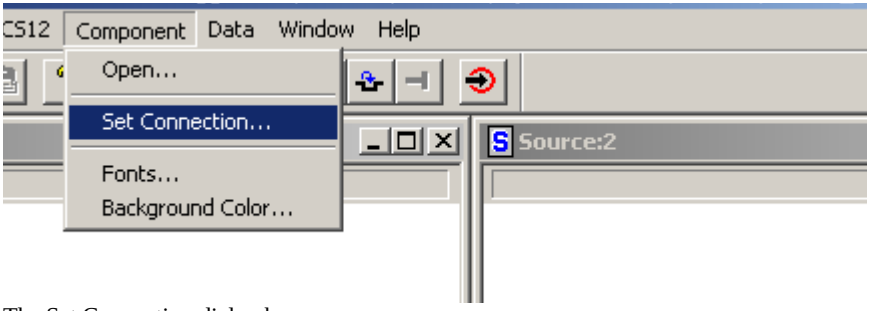
You have successfully switched connections to the FCS connection. The values and use of each HC08 FCS menu entry is explained in the Full Chip Simulation chapter of this manual.

Loading the P&E Multilink/Cyclone Pro Connection

To load the Multilink/Cyclone Pro connection from within an existing project, take the following steps:

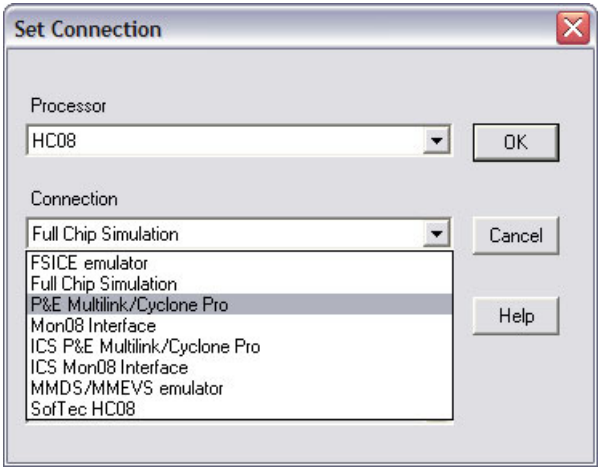
1. From the Debugger main menu, select *Component* | *Set Connection...*, as shown below.

Figure 10.11 Component Menu



The Set Connection dialog box now appears.

Figure 10.12 Set Connection Dialog Box - Connection Menu



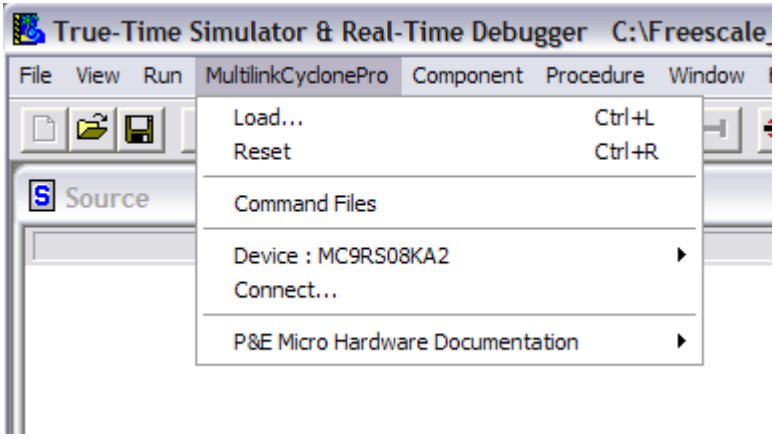
2. Within the Set Connection dialog box, press the Down Arrow button next to the Connection list box to display the list of available connections.

3. Select P&E Multilink/Cyclone Pro.

The Connection menu selection *P&E Multilink/Cyclone Pro* loads the proper drivers, etc. for the connection.

4. In the Debugger Main window, the Connection heading has been renamed **MultilinkCyclonePro**. Click on this heading to display its menu and list of selections.

Figure 10.13 MultilinkCyclone Pro Menu

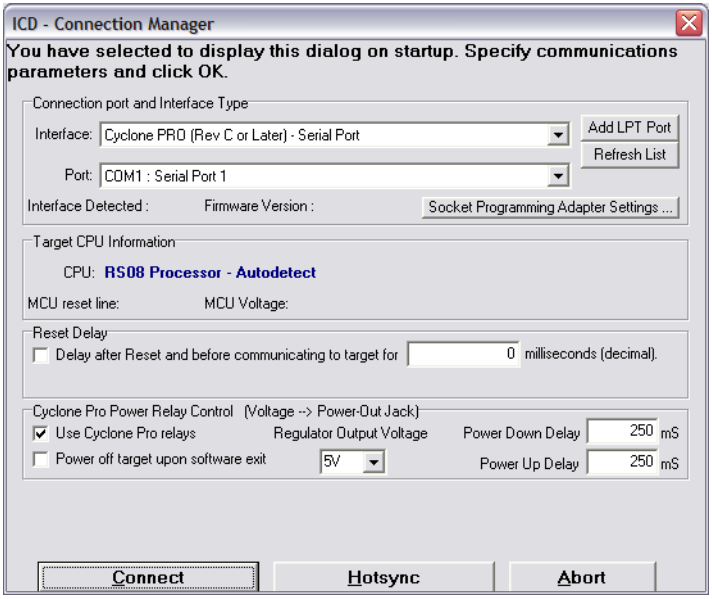


The menu selection *MultilinkCyclonePro* | *Load...* loads an executable ("`.abs`") file into connection memory. The file's program counter points to the first instruction of the startup section.

The menu selection *MultilinkCyclonePro*| *Reset* triggers a reset of the connection and executes the command file "`reset.cmd`".

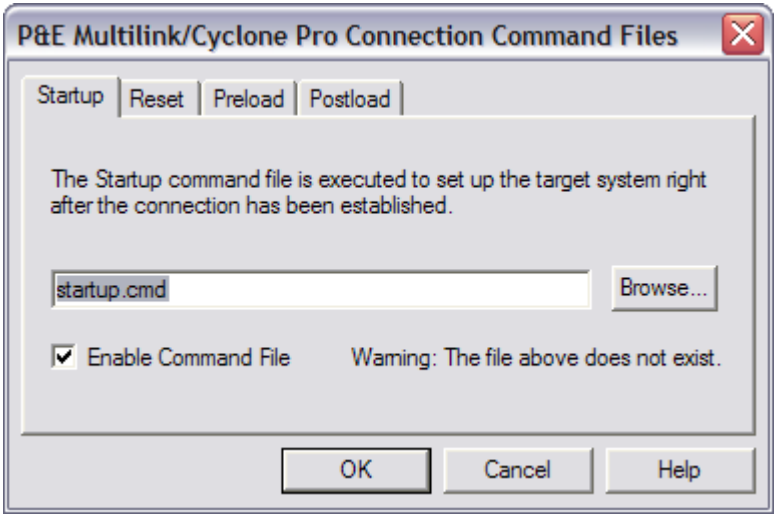
The menu selection *MultilinkCyclonePro* | *Connect...* takes you to the P&E ICD connection manager dialog box.

Figure 10.14 P&E ICD Connection Manager Dialog Box



The menu selection *MultilinkCyclonePro* | *Command Files...* takes you to the Command Files dialog box.

Figure 10.15 Command Files Window



11

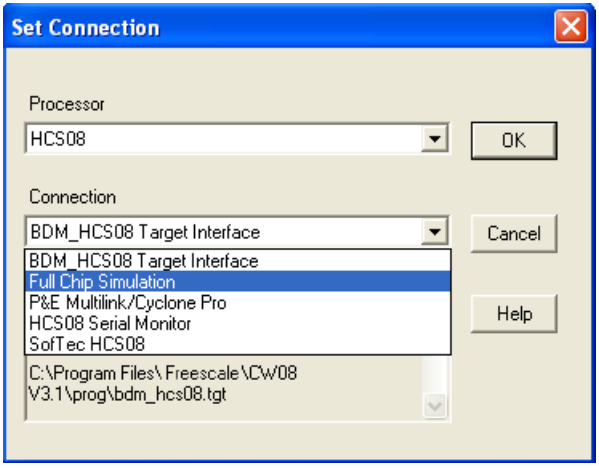
HC08 Full Chip Simulation

Full Chip Simulation (FCS) connection runs a complete simulation of all processor peripherals and I/O on the user's Personal Computer. Because of this it does not require an MCU development board to be connected to your PC. Each derivative has a totally different simulation engine to accurately simulate the memory ranges, I/O, and peripherals for a given derivative (for more information on selecting a specific derivative, please see the "Select Device" section below.

Configuration Procedure

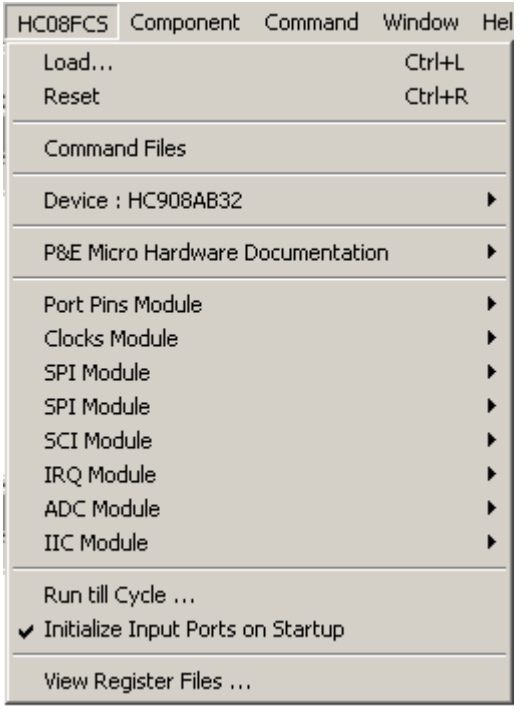
Choose the Full Chip Simulation option from the Set Connection dialog box. See Figure 11.1 on page 249.

Figure 11.1 Set Connection Dialog Box - Full Chip Simulation Option



When you have selected this option, the Connection dropdown menu becomes the HC08FCS Menu, and appears as shown in Figure 11.2 on page 250.

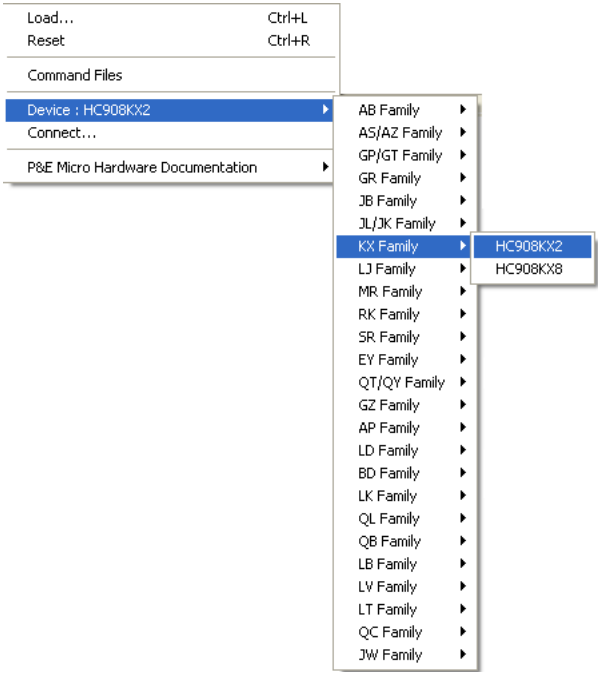
Figure 11.2 Connection (HC08FCS) Dropdown Menu



Select Device Option

The Device option on the HC08FCS menu allows the user to select the particular Freescale processor that they wish to use. When choosing Device from the HC08FCS menu, additional extended menus open which allow you to select the family (e.g. KX Family), and device type (e.g. HC908KX2) of the MCU that you are using. See Figure 11.3 on page 251.

Figure 11.3 Device Option Extended Menus



Full Chip Simulation Module Commands

The HC08FCS Menu contains the Full Chip Simulation commands for the modules that have specialty commands associated with them for a chosen device. For more information about specific module commands, please refer to the Full Chip Simulation section describing the particular module.

Run Till Cycle Option

The Run Till Cycle command lets you begin execution of code, and stop execution when the specified cycle count is reached. Please note that the parameter given is not the number of cycles that will be executed, but rather the total cycle count of the simulator (displayed in the Register Window).

This command is extremely useful for verifying specific timings of a given event; running until a given event is complete, or just before it completes to enable stepping through the event itself; or any application where cycle-timed execution is desired.

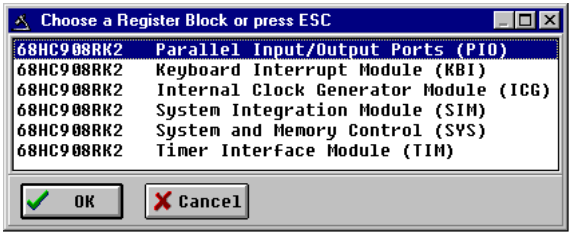
Initialize Input Ports On Startup

The Initiailze Input Ports on Startup option will initialize all simulated inputs to "$00" when the software is started, or when the Device Mode or Debug Target is switched. This initialization doesn't apply to a **reset** command. When this option is changed, the new state will take effect the next time HiWave is started (or Device Mode/Debug Target is switched).

View Register Files Option

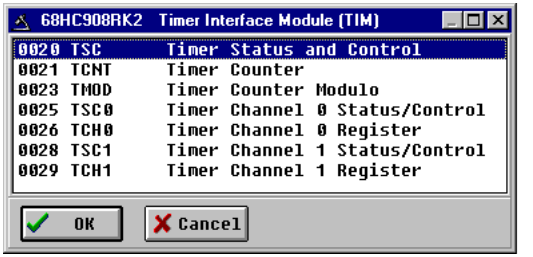
The “View Register Files...” option in the HC08FCS Menu also gives the user the option of running the register file viewer/editor. If register files are available for the device that you have chosen, the “Choose a Register Block” dialog box (see Figure 11.4 on page 252) is opened. You may also open it by entering the R command in the Command Window command line.

Figure 11.4 Choose A Register Block Dialog Box


Choose a Register Block or press ESC
68HC908RK2 Parallel Input/Output Ports (PIO)
68HC908RK2 Keyboard Interrupt Module (KBI)
68HC908RK2 Internal Clock Generator Module (ICG)
68HC908RK2 System Integration Module (SIM)
68HC908RK2 System and Memory Control (SYS)
68HC908RK2 Timer Interface Module (TIM)
OK
Cancel

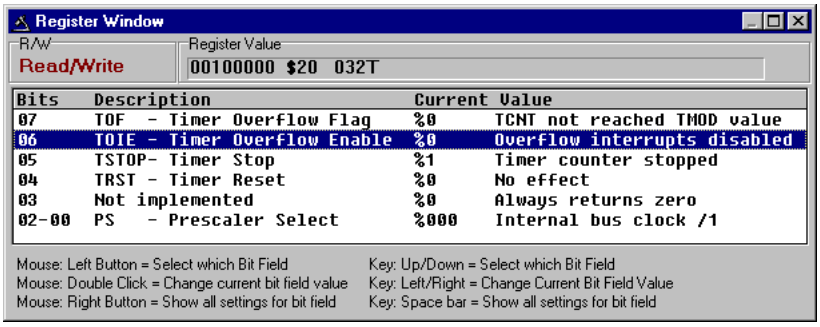
If register files have been installed on the host computer, selecting a block brings up the Register Block register listing in the Timer Interface Module dialog box (see Figure 11.5 on page 252), which shows a list of the associated registers, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.

Figure 11.5 Timer Interface Module Dialog Box - Register Listing


68HC908RK2 Timer Interface Module (TIM)
0020 TSC Timer Status and Control
0021 TCNT Timer Counter
0023 TMOD Timer Counter Modulo
0025 TSC0 Timer Channel 0 Status/Control
0026 TCH0 Timer Channel 0 Register
0028 TSC1 Timer Channel 1 Status/Control
0029 TCH1 Timer Channel 1 Register
OK
Cancel

Selecting a file in this dialog box brings up the Register Window (see Figure 11.6 on page 253), which displays the values and significance for each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.

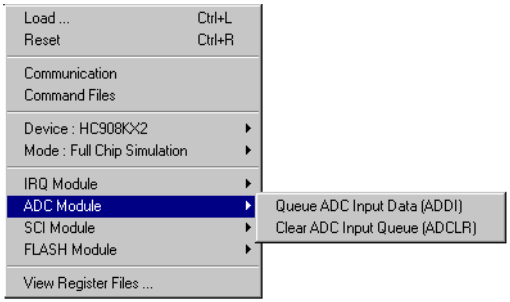
Figure 11.6 Register Window


Register Window
R/W
Read/Write
Register Value
00100000 $20 032T
Bits Description Current Value
07 TOF - Timer Overflow Flag %0 TCNT not reached TMOD value
06 TOIE - Timer Overflow Enable %0 Overflow interrupts disabled
05 TSTOP- Timer Stop %1 Timer counter stopped
04 TRST - Timer Reset %0 No effect
03 Not implemented %0 Always returns zero
02-00 PS - Prescaler Select %000 Internal bus clock /1
Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

Peripheral Modules Options

If you have selected a device (see Select Device Option), the HC08FCS Connection Menu displays a list of peripherals (Modules)for the device you have chosen. The Module's associated commands appear as extended menus .

Figure 11.7 HC08FCS Menu - ADC Module Extended Menu Options



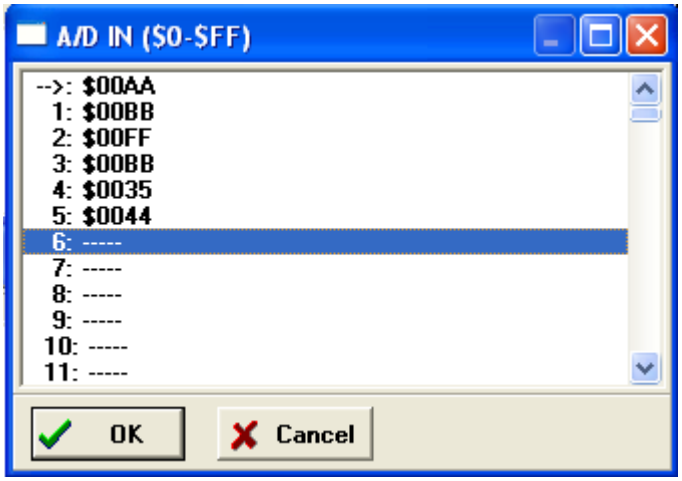
Placing your mouse over a peripheral opens an extended menu which lists its associated commands. Click on a command in order to execute that command.

ADC Module Option

In Full Chip Simulation Mode, this option lets you simulate all the functionality of the Analog to Digital Conversion (ADC) module including data input on all ADC channels,

flag polling, interrupt operation as well as the bus and CGMXCLK reference clock sources. Full Chip Simulation (FCS) mode uses the buffered input structure to simulate the ADC inputs. The user can queue up to 256 data values. To queue the ADC Input Data, the user should use the ADDI command in the command prompt. If the data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is provided, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the ADC. The conversion takes place after a proper value is written to the ADC Status and Control register. Once the conversion occurs, the arrow moves to the next value in the ADC Buffer.

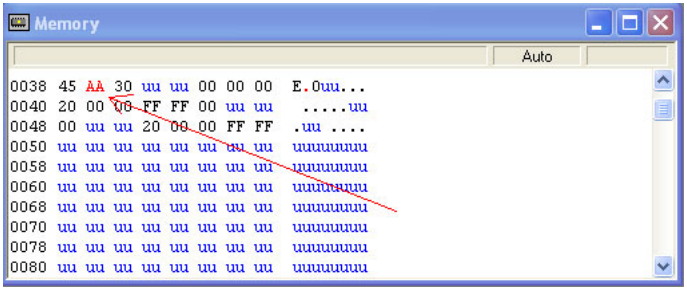
Figure 11.8 ADC IN Buffer Display



At any point, the ADCLR command can be used to flush the input buffer for the ADC simulation.

After the conversion is complete, the first queued value is passed from the data buffer into the ADC data register. It can be observed in the memory window by displaying the memory location corresponding to the ADC data register.

Figure 11.9 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

When the conversion is complete, the appropriate flag will be set. If interrupts are enabled, the Program Counter will change flow to the interrupt routine (as defined in the vector space of the MCU). For more information on ADC configuration, please refer to the Freescale Manual corresponding to the microprocessor that you are using.

ADC User Commands

ADDI Command

The ADDI command allows the user to input the data into the ADC converter. If a data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next to the ADC. The maximum number of input values is 256 bytes.

Syntax:

```
>ADDI [<n>]
```

Where:

<n> The value to be entered into the next location in the input buffer.

Example:

```
>ADDI $55
```

Set the next input value to the ADDI to $55

```
>ADDI
```

Pull up the data window with all the input values.

ADCLR Command

The ADCLR command can be used to flush the input buffer for ADC simulation. This will reset the input data buffer and clear out all values. Notice that if the ADC is currently

using a value, this command will not prevent the ADC from using it. See ADDI Command for information on how to access the input buffer of the ADC interface.

Syntax:

```
>ADCLR
```

Example:

```
>ADCLR
```

Clear the input buffer for ADC simulation.

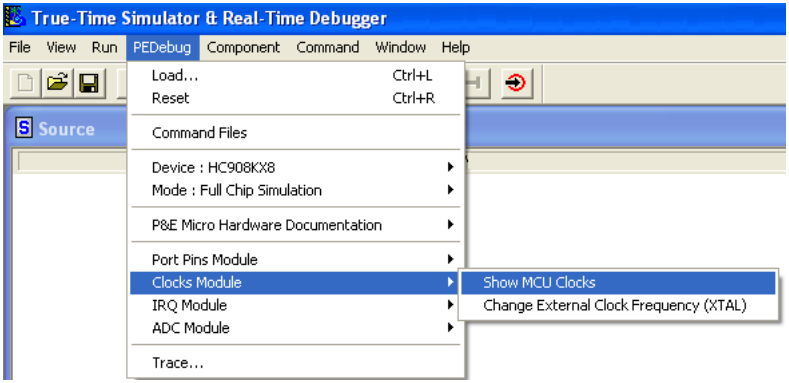
Clock Generation Module Option

In Full Chip Simulation Mode, this option lets you simulate all functionality of the Clock Generation Module (CGM), including:

- Phase Locked Loop (PLL) generation
- Automatic lock detection
- Interrupt
- Acquisition
- Tracking
- Flag polling

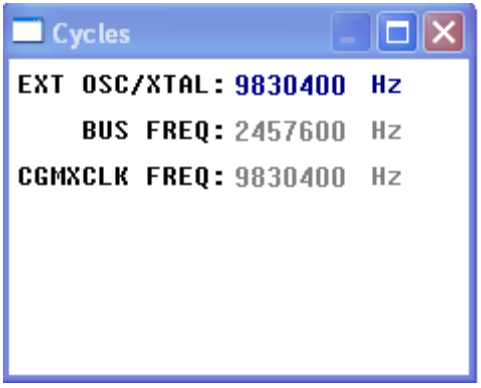
Full Chip Simulation mode uses simulated External Oscillator Frequency change command (XTAL) to allow the user to input the desired XTAL value. To check the current value of the External Oscillator, Bus Frequency and CGMXCLK Frequency, the user should open the HC08FCS menu, then select Clocks Module -> Show MCU Clocks.

Figure 11.10 Show MCU Clocks Menu



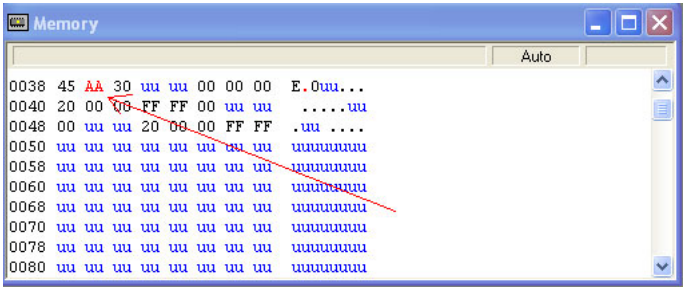
Once the MCU Clocks Menu is selected, the Cycles window containing all of the above-mentioned Clock Frequencies will be displayed.

Figure 11.11 Frequency Display



Once the CGM is properly configured, the user can monitor the status of the PLL by polling the corresponding flag. If PLL interrupt is enabled, Full Chip Simulation will jump to an appropriate subroutine as long as the interrupt vector is properly defined. To observe the flag going up as a result of the corresponding CPU event, situate your Memory Window on the memory location of the CGM Status and Control register.

Figure 11.12 Memory Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

For more information on how to properly configure Clock Generation, please refer to the conversioncale reference manual corresponding to the microprocessor that you are using.

CGM Commands

XTAL Command

The XTAL command is used to change the value of the simulated external oscillator. This in turn will affect the input to the PLL/DCO, and therefore the bus frequency. The P&E simulator is a cycle-based simulator, so changing the XTAL value will not affect the speed of simulation; it will, however, affect the ratio in which peripherals receive cycles. Certain peripherals which run directly from the XTAL will run at different speeds than those that run from the bus clock.

Syntax:

```
>XTAL <n>
```

Where:

n, by default, is a hexadecimal number, representing the simulated frequency of an external oscillator. Adding the suffix “t” to the n parameter will force the input value to be interpreted as base 10.

Example:

```
>XTAL
```

Brings up an input window. The default base for this input value is 10. However, this value can be forced to a hexadecimal format through use of the suffix “h”.

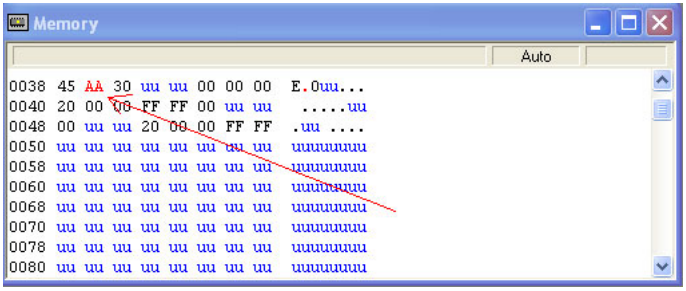
High Resolution PWM Module

In Full Chip Simulation Mode, this option lets you simulate all functionality of the High Resolution PWM (HRP) module, including:

- pulse width modulation
- flag polling
- interrupt enabled mode of operation
- variable period and dead time insertion
- shutdown input for fast disabling of outputs

To ensure that the HRP firmware is functioning as expected, the user can observe the toggling of I/O pins that are multiplexed with the HRP module. In order to do that, the HRP module has to be configured for the desired period with the enabled output on corresponding TOP and BOT pins. The Memory Window should display the address of the register corresponding to the appropriate I/O port.

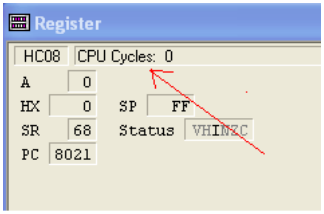
Figure 11.13 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

If the HRP module has the Shutdown pin enabled, the SHTIF Flag can be observed via the Memory window once the Shutdown event takes place. If the HRP interrupt is enabled, the FCS will jump to an appropriate subroutine as long as the HRP interrupt vector is properly defined. To observe the SHTDWN Interrupt Flag going up as a result of the corresponding CPU event, situate your Memory Window on the memory location of the HRP Control register.

To observe the accuracy of the HRP module operation, the user can observe the number of CPU cycles that it takes for the event to occur. The cycle counter is only incremented as the user steps through the code. To determine the exact amount of cycles over which the event occurs, one can either observe the cycle display in the Register Window or use the built in simulation commands. To display the current number of cycles in the Command window, the CYCLES command should be used. To change the number of cycles in the cycle counter, use CYCLES <n>, where <n> is the new cycle value. If the number of cycles of which an event should consist has already been pre-calculated, one can use CYCLE 00 to reset the number of cycles and GOTOCYCLE <n> to run through the code until the place where expected event should take place.

Figure 11.14 Register Window With Cycles Display



High Resolution PWM Commands

CYCLES Command

The CYCLES command changes the value of the cycles counter. The cycles counter counts the number of the processor cycles that have passed during execution. The Cycles Window shows the cycle counter. The cycle count can be useful for timing procedures.

Syntax:

```
>Cycles <n>
```

Where:

<n> Integer value for the cycles counter

Examples:

```
>CYCLES 0
```

Reset cycles counter

```
>CYCLES 1000
```

Set cycle counter to 1000.

GOTOCYCLE Command

The GOTOCYCLE command executes the program in the simulator beginning at the address in the program counter (PC). Execution continues until the cycle counter is equal to or greater than the specified value, until a key or the Stop button on the toolbar is pressed, until it reaches a break point, or until an error occurs.

Syntax:

```
>GOTOCYCLE <n>
```

Where:

<n> Cycle-counter value at which the execution stops

Example:

```
>GOTOCYCLE 100
```

Execute the program until the cycle counter equal 100.

SHTDWN Command

When the Shutdown pin is enabled in the High Resolution PWM Control Register (HRPCTRL), this command can be used to change the state of the SHTDWN pin. The Shutdown pin will then take on this state after the next step is executed in the simulator. Note that if interrupts are enabled, issuing a SHTDWN 0 command will trigger an interrupt that will not be cleared until the SHTIF bit is cleared in the HRPCTRL and a SHUTDWN 1 command is issued. Please note that this interrupt is both edge and level sensitive.

If the SHTDWN pin is not enabled in the HRPCTRL, this command will have no effect.

Syntax:

```
>SHTDWN <n>
```

Where:

<n> = 1 or 0, representing the state of the SHTDWN pin on the next simulated clock cycle.

Example:

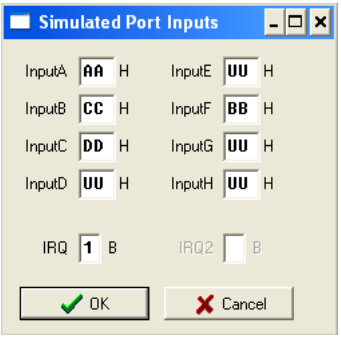
```
>SHTDWN 0
```

Change the state of the SHTDWN pin to 0.

Input/Output (I/O) Ports Module

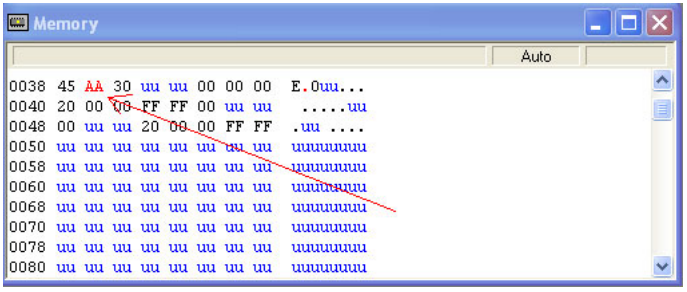
In Full Chip Simulation Mode, this option lets you simulate all input and output functionality of the Input/Output (I/O) Ports module. The Full Chip Simulation (FCS) mode uses a set of designated commands to simulate the input and output activity on corresponding I/O port pins. To define an input state of the specific port one should write the INPUT <x> <n> command in the Command line window. The <x> represents corresponding I/O port, while the <n> stands for the input value that should be written to this port. At the same time, the INPUTS command can be used to bring up the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports. Please see the documentation for INPUT<x> Command and **INPUTS Command** for more information about the various forms of this command.

Figure 11.15 Simulated Port Inputs Dialog Box



Using the Simulated Port Inputs dialog box, the input value to any I/O port can be reconfigured. The INPUTS command can be used to reconfigure the output values on any relevant I/O port. The manipulation of I/O port pins can be observed in the Memory Window.

Figure 11.16 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

Please note that if the regular I/O pins are multiplexed to be used by a different MCU Module, they might not be available for general I/O functionality. For more information

on how to properly configure I/O pins, please refer to the Freescale user manual corresponding to the microprocessor that you are using.

Input/Output Ports Commands

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax:

```
>INPUT<x>  <n>
```

Where:

<x> is the letter representing corresponding port

<n> Eight-bit simulated value for port <x>

Example:

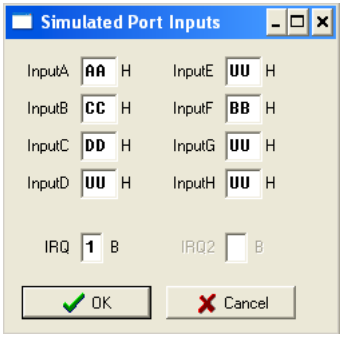
```
>INPUTA AA
```

Simulate the input AA on port A.

INPUTS Command

In Full Chip Simulation and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 11.17 on page 263 below. The user may then use this box to specify the input states of port pins and IRQ.

Figure 11.17 Simulated Port Inputs Dialog Box



When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax:

```
>INPUTS
```

Example:

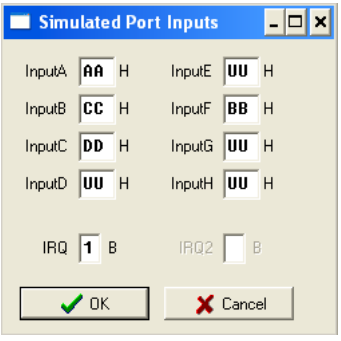
```
>INPUTS
```

Show I/O port input values.

External Interrupt Module

In Full Chip Simulation Mode, this option lets you simulate the input, flag polling and interrupt functionality of the External Interrupt (IRQ) module. The Full Chip Simulation Mode uses the INPUTS command to let the user monitor and change the simulated value of the IRQ input pin state. Please see the documentation for INPUT<x> Command and **INPUTS Command** for more information about the various forms of this command. Once the user enters the INPUTS command into the command line prompt, the Simulated Port Inputs window will come up. In addition, the state of the IRQ pin can be modified directly using the IRQ <n> command (documented below).

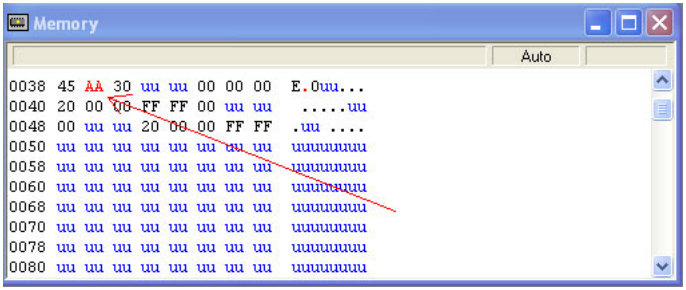
Figure 11.18 Simulated Port Inputs Dialog Box



After the IRQ event occurs, the appropriate flag will be set in the corresponding IRQ register. The user is able to poll the IRQ flag if the Polling Mode is simulated. In the Interrupt Mode, the simulator will branch to an appropriate interrupt subroutine as long as the IRQ interrupt vector is properly configured. For more information on IRQ configuration, please refer to the Freescale user manual corresponding to the microprocessor that you are using.

Following the IRQ event, the IRQ Flag can be observed going up in the IRQ Status and Control register.

Figure 11.19 Memory Component Window

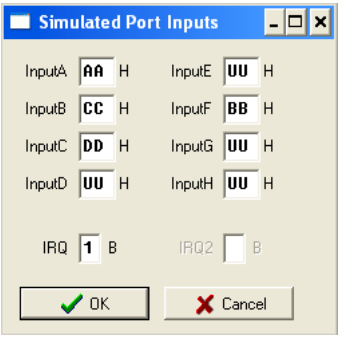

Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

IRQ Commands

INPUTS Command

In Full Chip Simulation and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 11.20 on page 265. The user may then use this box to specify the input states of port pins and IRQ.

Figure 11.20 Simulated Port Inputs Dialog Box



When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax:

```
>INPUTS
```

Example:

```
>INPUTS
```

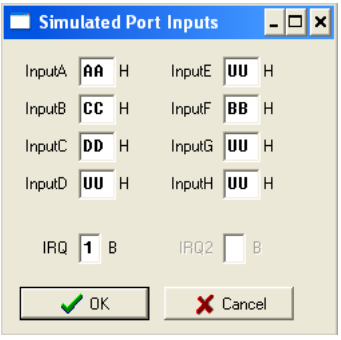
Show I/O port input values.

NOTE The IRQ pin state can be directly manipulated with IRQ command. For example, "IRQ 1" will simulate a high state on the IRQ pin; likewise, "IRQ 0" will simulate a logic-low state on the IRQ pin.

Keyboard Interrupt Module

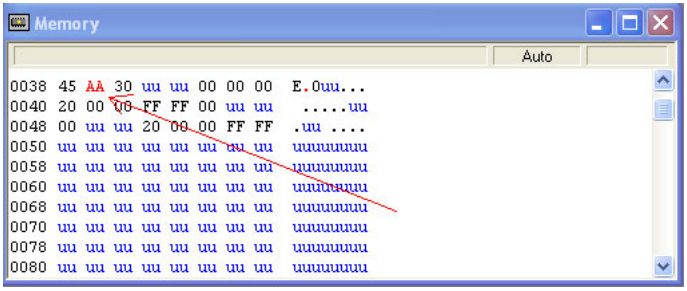
In Full Chip Simulation Mode, this module will simulate all functionality of the Keyboard Interrupt (KBI) module including the edge-only, edge and level interrupt, and flag polling modes of operation. The Full Chip Simulation (FCS) mode uses simulated port inputs to trigger the KBI event from the proper I/O port pin. To define an input state of the specific port one should write the INPUT<x> <n> command in the Command line window. The <x> represents the corresponding I/O port, while <n> stands for the input value that should be written to this port. At the same time, the INPUTS command can be used to bring up the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports. Please see the documentation for **INPUT<x> Command** and INPUTS Command for more information about the various forms of this command.

Figure 11.21 Simulated Port Inputs Dialog Box



Using the Simulated Port Inputs Dialog, the input value to any I/O port can be reconfigured. In order to trigger the event, the inputs to the port should be manipulated in the appropriate manner, depending on whether the KBI is configured for edge-only or edge and level. Once the KBI event takes place, the KEYF Flag bit, which is a part of the Keyboard Status and Control register, can be observed in the Memory Window.

Figure 11.22 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 06 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

The user is able to poll the KBI Interrupt Pending flag if the polling Mode is simulated. In Interrupt Mode, the simulator will branch to an appropriate interrupt subroutine as long as the KBI interrupt vector is properly configured. For more information on KBI configuration, please refer to the Freescale user manual corresponding to the microprocessor that you are using.

Keyboard Interrupt Commands

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax:

```
>INPUT<x>  <n>
```

Where:

<x> is the letter representing corresponding port

<n> Eight-bit simulated value for port <x>

Example:

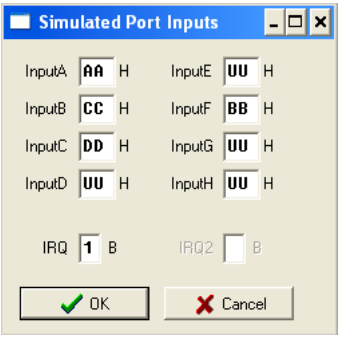
```
>INPUTA AA
```

Simulate the input AA on port A.

INPUTS Command

In Full Chip Simulation and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 11.23 on page 268. The use may then use this box to specify the input states of port pins and IRQ.

Figure 11.23 Simulated Port Inputs Dialog Box



When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax:

```
>INPUTS
```

Example:

```
>INPUTS
```

Show I/O port input values.

Multi Master Inter-Integrated Circuit Module

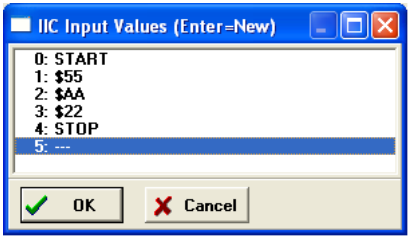
In Full Chip Simulation Mode, this module will simulate all functionality of the Multi-Master Inter-Integrated Circuit (MMIIC) module including:

- Flag polling
- Interrupt enabled mode
- Transmission and reception of external data
- Master and slave modes of operation
- START and STOP signal generation detection
- Acknowledge bit generation detection

The FCS mode uses the buffered input/output structure to simulate MMIIC inputs. The user can queue up to 256 data bytes into the input buffer. The output buffer of the USB module can also hold 256 output bytes. To queue the MMIIC Input Packets, one should use IICDI <...> command in the command prompt. For a more detailed description of the command, please refer to the MMIIC Commands section. If the MMIIC packet

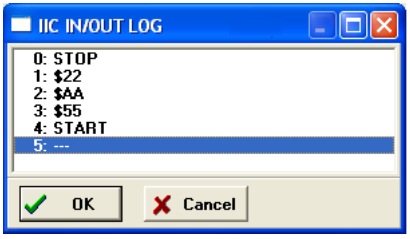
parameters are properly defined, the packet is placed into the next slot in the input buffer. Otherwise, if no parameters are provided, an MMIIC Input Buffer window is displayed. Different MMIIC packet parameters can be entered while the window is open including START, STOP, ACK, NACK and data bytes. An arrow points to the byte that will be used next as input to the MMIIC. The data from the MMIIC input buffer is written to the MMIIC module registers once the MMIIC module is turned on and properly configured for receiving data from an external MMIIC device. Once the simulation of the data transmission is over, the arrow moves to the next value in the MMIIC Input Buffer.

Figure 11.24 IIC Input Buffer Display



The MMIIC data input/output log buffer simulation allows the user to gain access to the past 256 MMIIC data bytes that have been shifted in and out of the module. To bring up the IIC IN/OUT LOG buffer dialog, the IICDO command should be used.

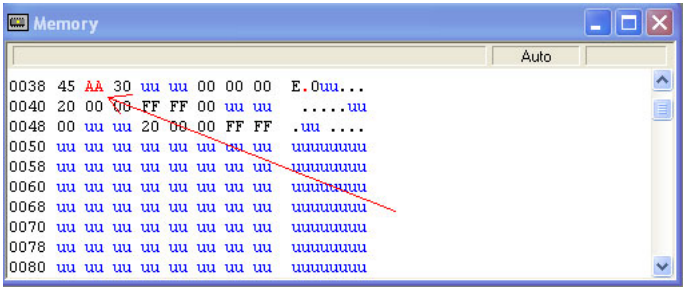
Figure 11.25 IIC IN/OUT LOG Buffer Display



At any point, IICCLR command can flush the input as well as input/output log MMIIC buffers.

After the MMIIC simulated input is received, the first queued in data byte is passed from the data buffer into the corresponding MMIIC module registers. It can be observed in the Memory Window by displaying the appropriate register location there.

Figure 11.26 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

The user can also observe different MMIIC flags in the Memory Window. If the module is run in Flag Polling mode, the user should poll the flag corresponding to the expected MMIIC event. If the MMIIC interrupts are enabled, the FCS will jump to an appropriate subroutine as long as the MMIIC interrupt vectors are properly defined.

For more information on how to configure MMIIC module for desired operation, refer to the Freescale user manual corresponding to the microprocessor that you are using.

Multi-Master Inter-Integrated Circuit Module Commands

IICDI Command

The IICDI command allows the user to input data into a buffer of data which will be shifted into the MMIIC module when it receives data from an external device. If a data parameter is given, the value is placed into the next slot in the input buffer. Otherwise if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. The maximum number of input values is 256.

This command is useful for either inputting response data from a slave target or for inputting data packets from an external master. Note that when the microprocessor attempts to read an acknowledge from an external device, and the next value in the buffer is neither ACK nor NACK, the microprocessor automatically receives an ACK signal (i.e. assumes ACK unless NACK is specified).

Syntax:

```
>IICDI [<n>][START][STOP][ACK][NACK]
```

Where:

<n> The value to be entered into the next location in the input buffer

START indicates the incoming START signal

STOP indicates the incoming STOP signal

ACK corresponds to ACK signal

NACK corresponds to NACK signal

For a detailed description of the IIC protocol and a proper way IIC module should be configured, please refer to the Freescale user manual corresponding to the Freescale microprocessor that you are using.

Example:

```
>IICDI
```

Pulls up the data window with all the input values

```
>IICDI 22 33
```

This is an example of data being returned from a slave device. Once the MCU transmits a start signal and the target address, it receives and ACK from the slave device. An ACK is implied unless a NACK is specified via the IICDI command. The next tow data bytes read will be 22 and 23. If the microprocessor attempts to read another byte, it will get an $FF value followed by a NACK signal (NACK because nothing remains in the input buffer). The receiving device will then generate a STOP signal. A more exact input from a device designed to return two bytes is:

```
>IICDI ACK 22 ACK 23 NACK
```

MMIIC in master mode transmits to a slave. If the slave device acknowledges all output bytes of the transmitting device, there is no need to specify an input packet. If the master device were going to transmit an address and two bytes, the following packet would be equivalent to no packet:

```
>IICDI ACK ACK ACK
```

If, however, the slave receiver is designed to generate a NACK signal after the second received data byte, the proper response packet is:

```
>IICDI ACK ACK NACK
```

The address result being the first ACK, the first data result being the second ACK, and the second data byte being the NACK.

MMIIC in MASTER mode is not acknowledged by any Slave:

```
>IICDI NACK
```

If the NACK signal is entered before our master device transmits a START signal, then the master device will get back a NACK when it tries to read an acknowledge after the address is outputted. The master device will then generate a STOP signal and release the BUS.

MMIIC in SLAVE mode receives a Write from an external Master:

This example is for an external master which is writing to the microprocessor configured to simulate the slave mode operation. The packet contains both START and STOP signals which will put the simulated device into the slave mode.

```
>IICDI START 55 AA 22 STOP
```

This input should add 5 values to the input queue which is a packet from an external master including the following procedure steps:

1. A start signal comes in.
2. The address $55 comes in specifying a write (slave receive); The current simulated device should have previously set its Address Register to $55.
3. The data byte $AA comes in.
4. The data byte $22 comes in.
5. A STOP signal comes in.

IICDO

The IICDO command displays a window, which shows data shifted in as well as shifted out of the IIC peripheral. An arrow is used to point to the last output value transmitted/ received. The maximum number of output values that the buffer can hold is 256.

Syntax:

```
>IICDO
```

Example:

```
IICDO
```

View data from the input/output log buffer for IIC simulation.

IICCLR

The IICCLR command can be used to flush the input and output buffers for MMIIC simulation. This will reset the buffers and clear all values. Notice that if the MMIIC is currently shifting a value, this command will not prevent the MMIIC from finishing the transfer.

Syntax:

```
>IICCLR
```

Example:

```
>IICCLR
```

Clear input and output buffers for IIC simulation.

FCSMSCAN Controller Module

The MSCAN Controller Module fully simulates the operation of the MSCAN08 Protocol Version 2.0 based device, including:

- Flag polling
- Interrupt enabled mode
- 0-8 bytes data length
- Transmission and reception of external data

FCSMSCAN08 Background and Assembly Example

The MSCAN08 peripheral is a scalable control area network (CAN) 2.0 compliant device that allows microcontrollers to exchange data between themselves at high speeds. This is done through a high-speed serial link that is deterministic and reliable. CAN devices are often utilized in automobiles, where multiple microcontrollers need to be connected into a network. The CAN specification indicates that any unit on the bus can be a master at any time, where they can send a message to another unit whenever they wish if the bus is free to do so. All of these messages can be setup through the CAN I/O commands built into the simulator. This section goes through an example of this, showing how the simulator can be used to test out code for driving the CAN peripheral

Listing 11.1 on page 274 is an example assembly program that exercises the CAN 2.0 peripheral on the 68HC908AZ60. The source assembly file can be download from P&E Microcomputer website at www.pemicro.com. With minor modifications this assembly file can be added and compiled as a part of your Freescale project. It initializes the CAN peripheral on the microcontroller for operation and uses interrupts from the CAN peripheral to service requests from other controllers. Once the code is compiled and loaded into the microprocessor that you are using, simulator commands can be used to emulate the CAN messages that come from the host. Let us first break down what is being done in the example program.

FCSDemo CAN Program

This code simply sets up the CAN controller to be ready to receive CAN messages intended for hexadecimal address 0x01234567. Note that this peripheral allows the user to setup filters for address matching, where only certain bit fields of the address need to match in order to receive the data. In our case we just setup the filters for 32-bit exact match. An interrupt service routine is created for the CAN peripheral and is used to collect messages that are received.

The code also tries to send out three fixed messages in memory out to the network. Each of these messages has three different locations that they are sent to. The priorities for the

messages are also setup to be different, to show how prioritization works with the CAN peripheral.

Listing 11.1 68HC908AZ60 MSCAN Demonstration Application

```
; 68HC908AZ60 MSCAN Demonstration Application
; (C)opyright P&E Microcomputer Systems, 2000

$pagewidth 120t

RAMStart       equ    $0050              ; start of RAM for AZ60
RomStart       equ    $8000              ; start of Flash1 for AZ60
VectorStart    equ    $FFCC              ; start of Vectors for AZ60

$Include 'az60regs.inc'

CANAddress    equ     $01234567     ; fixed address of this CAN Module

    org RamStart
buffptr       ds      2         ; pointer into the local data buffer
dataptr       ds      2         ; pointer into the CAN receiver data
buffer
datacount     ds      1        ; holds the number of bytes in messages
PacketData    ds      $100     ; buffer for data from received packets
PacketDataEnd:

    org RomStart
* Transmission Message Descriptors
* All messages are extended data types
* Format is Address, Priority, Length, Data
Msg1_Desc:
       dw  $0001               ; Message 1 Address
       dw  $2345               ; Message 1 Address
       db  $03                 ; Message 1 Priority
       db  $08                 ; Message 1 Length
       db  $11                 ; Data Byte 1
       db  $22                 ; Data Byte 2
       db  $33                 ; Data Byte 3
       db  $44                 ; Data Byte 4
       db  $55                 ; Data Byte 5
       db  $66                 ; Data Byte 6
       db  $77                 ; Data Byte 7
       db  $88                 ; Data Byte 8
M1Desc_End:

Msg2_Desc:
       dw  $0002               ; Message 2 Address
       dw  $3456               ; Message 2 Address
```

```
       db  $02                  ; Message 2 Priority
       db  $04                  ; Message 2 Length
       db  $12                  ; Data Byte 1
       db  $34                  ; Data Byte 2
       db  $56                  ; Data Byte 3
       db  $78                  ; Data Byte 4
M2Desc_End:

Msg3_Desc:
       dw  $0003                ; Message 3 Address
       dw  $4567                ; Message 3 Address
       db  $01                  ; Message 3 Priority
       db  $02                  ; Message 3 Length
       db  $55                  ; Data Byte 1
       db  $AA                  ; Data Byte 2
M3Desc_End:

************************************************************
* Init_CAN - The CAN is placed into the soft reset state,     *
*            where the control and timing registers can be    *
*            set and the identifier and mask registers can be *
*            configured. After this, the module is placed in  *
*            normal mode in order to synchronize with the CAN *
*            bus.                                             *
************************************************************
Init_CAN:
       lda   #$01
       sta   CMCR0              ; place MSCAN08 into soft reset state
       lda   #$01
       sta   CMCR1              ; CAN clock source = CGMOUT*2
       lda   #$01
      sta   CBTR0             ; set SJW=0, baud rate prescalar=div by 2
       lda   #$27
      sta   CBTR1              ; set TSEG1=7 (8Tq), TSEG2=2 (3Tq), one
sample per bit
       lda   #{(CANAddress>21t) & $FF} ; set identifier acceptance
register to CAN address
       sta   CIDAR0
       lda   #{(CANAddress & $38000)>15t}
       ora   #{(CANAddress & $1C0000)>13t}
       ora   #$18
       sta   CIDAR1
       lda   #{(CANAddress & $7F80)>7t}
       sta   CIDAR2
       lda   #{(CANAddress & $7F)<1t}
       sta   CIDAR3
       lda   #$00
```

```
       sta   CIDMR0             ; set identifier mask register to exact
match
       sta   CIDMR1
       sta   CIDMR2
       sta   CIDMR3
       lda   #00
       sta   CIDAC              ; set identifier acceptance for single
32-bit filter
       ldhx  #Msg1_Desc         ; set up transmission message 1
       lda   0,x                ; get message address
       lsla
       lsla
       lsla
       sta   CT0IDR0
       lda   1,x                ; get next byte in message address
       and   #$E0
       lsra
       lsra
       lsra
       lsra
       lsra
       ora   CT0IDR0
       sta   CT0IDR0
       lda   1,x                ; get next byte in message address
       and   #$1C
       lsla
       lsla
       lsla
       ora   #$18
       sta   CT0IDR1
       lda   1,x                ; get next byte in message address
       and   #$03
       lsla
       ora   CT0IDR1
       sta   CT0IDR1
       lda   2,x                ; get next byte in message address
       and   #$80
       rola
       rola
       ora   CT0IDR1
       sta   CT0IDR1
       lda   2,x                ; get next byte in message address
       and   #$7F
       lsla
       sta   CT0IDR2
       lda   3,x                ; get next byte in message address
       and   #$80
       rola
```

```
rola
ora   CT0IDR2
sta   CT0IDR2
lda   3,x                ; get next byte in message address
and   #$7F
lsla
sta   CT0IDR3
lda   4,x                ; get the message priority
sta   CT0TBPR
lda   5,x                ; get the message length
sta   CT0DLR
sta   datacount
lda   6,x                ; get the message data
sta   CT0DSR0
lda   7,x                ; get the message data
sta   CT0DSR1
lda   8,x                ; get the message data
sta   CT0DSR2
lda   9,x                ; get the message data
sta   CT0DSR3
lda   0a,x               ; get the message data
sta   CT0DSR4
lda   0b,x               ; get the message data
sta   CT0DSR5
lda   0c,x               ; get the message data
sta   CT0DSR6
lda   0d,x               ; get the message data
sta   CT0DSR7
ldhx  #Msg2_Desc         ; set up transmission message 2
lda   0,x                ; get message address
lsla
lsla
lsla
sta   CT1IDR0
lda   1,x                ; get next byte in message address
and   #$E0
lsra
lsra
lsra
lsra
lsra
ora   CT1IDR0
sta   CT1IDR0
lda   1,x                ; get next byte in message address
and   #$1C
lsla
lsla
lsla
```

```
        ora   #$18
        sta   CT1IDR1
        lda   1,x               ; get next byte in message address
        and   #$03
        lsla
        ora   CT1IDR1
        sta   CT1IDR1
        lda   2,x               ; get next byte in message address
        and   #$80
        rola
        rola
        ora   CT1IDR1
        sta   CT1IDR1
        lda   2,x               ; get next byte in message address
        and   #$7F
        lsla
        sta   CT1IDR2
        lda   3,x               ; get next byte in message address
        and   #$80
        rola
        rola
        ora   CT1IDR2
        sta   CT1IDR2
        lda   3,x               ; get next byte in message address
        and   #$7F
        lsla
        sta   CT1IDR3
        lda   4,x               ; get the message priority
        sta   CT1TBPR
        lda   5,x               ; get the message length
        sta   CT1DLR
        sta   datacount
        lda   6,x               ; get the message data
        sta   CT1DSR0
        lda   7,x               ; get the message data
        sta   CT1DSR1
        lda   8,x               ; get the message data
        sta   CT1DSR2
        lda   9,x               ; get the message data
        sta   CT1DSR3
        ldhx  #Msg3_Desc        ; set up transmission message 3
        lda   0,x               ; get message address
        lsla
        lsla
        lsla
        sta   CT2IDR0
        lda   1,x               ; get next byte in message address
        and   #$E0
```

```
        lsra
        lsra
        lsra
        lsra
        lsra
        ora   CT2IDR0
        sta   CT2IDR0
        lda   1,x               ; get next byte in message address
        and   #$1C
        lsla
        lsla
        lsla
        ora   #$18
        sta   CT2IDR1
        lda   1,x               ; get next byte in message address
        and   #$03
        lsla
        ora   CT2IDR1
        sta   CT2IDR1
        lda   2,x               ; get next byte in message address
        and   #$80
        rola
        rola
        ora   CT2IDR1
        sta   CT2IDR1
        lda   2,x               ; get next byte in message address
        and   #$7F
        lsla
        sta   CT2IDR2
        lda   3,x               ; get next byte in message address
        and   #$80
        rola
        rola
        ora   CT2IDR2
        sta   CT2IDR2
        lda   3,x               ; get next byte in message address
        and   #$7F
        lsla
        sta   CT2IDR3
        lda   4,x               ; get the message priority
        sta   CT2TBPR
        lda   5,x               ; get the message length
        sta   CT2DLR
        sta   datacount
        lda   6,x               ; get the message data
        sta   CT2DSR0
        lda   7,x               ; get the message data
        sta   CT2DSR1
```

```
SYNCHCAN:
       lda   #$00
       sta   CMCR0             ; allow MSCAN08 to synchronize to the bus
       lda   #$FF
       sta   CRFLG               ; Reset all CAN receiver flags
       lda   #01
       sta   CRIER               ; enable receiver full interrupt
       lda   #$07
       sta   CTFLG               ; Reset all CAN transmitter flags
       rts


*****************************************************************
* MAIN_INIT - This is the point where code starts executing    *
*              after a RESET.                                   *
*****************************************************************
MAIN_INIT:
       rsp
       ldhx  #PacketData        ; initialize buffer pointer to start of
buffer
       sthx  buffptr
       lda   #01
       sta   CONFIG1             ; disable COP watchdog
       lda   #01
       sta   CONFIG2             ; MSCAND=0 (enable MSCAN module)
       jsr   Init_CAN            ; Initialize CAN peripheral
       cli                       ; Allow interrupts to happen

main_loop:
       nop                       ; do nothing in main loop
       bra   main_loop

*****************************************************************
* CAN_ISR - CAN Interrupt Service Routine.                     *
*           Interrupts here if CAN packet is received that     *
*           has the address of this CAN module.                *
*****************************************************************
CAN_ISR:
       lda   CRDLR               ; get length of data in message
       cbeqa #0,DATADONE         ; if no data in message then done
       sta   datacount
       ldhx  #CRDSR0            ; initialize data pointer to first data
segment
       sthx  dataptr
STOREDATA:
       ldhx  dataptr             ; get data pointer to data in received
message
       lda   0,x                 ; load data from data segment
```

```
       aix   #1                  ; index pointer to next segment
       sthx  dataptr             ; save data pointer
       ldhx  buffptr             ; get pointer to local circular buffer
       sta   0,x                 ; store data in next location
       aix   #1                  ; index pointer to next location
       cphx  #PacketDataEnd      ; if end of circular buffer
       bne   CHECKDATAEND        ; then reset pointer to start of
circular buffer
       ldhx  #PacketData
CHECKDATAEND:
       sthx  buffptr
      dbnz  datacount,STOREDATA  ;if no more data in message then done
DATADONE:
       lda   #01
       sta   CRFLG               ; reset receive flag
       rti

*************************************************************
* DUMMY_ISR - Dummy Interrupt Service Routine.               *
*             Just does a return from interrupt.             *
*************************************************************
DUMMY_ISR:
       rti                        ; simple return

*************************************************************
* Vectors - Specifying Reset and MSCAN Interrupt Routines    *
*************************************************************
       org   VectorStart

       dw    dummy_isr           ; TIMA Channel 5 Vector
       dw    dummy_isr           ; TIMA Channel 4 Vector
       dw    dummy_isr           ; ADC Vector
       dw    dummy_isr           ; Keyboard Vector
       dw    dummy_isr           ; SCI Transmit Vector
       dw    dummy_isr           ; SCI Receive Vector
       dw    dummy_isr           ; SCI Error Vector
       dw    dummy_isr           ; CAN Transmit Vector
       dw    CAN_ISR             ; CAN Receive Vector
       dw    dummy_isr           ; CAN Error Vector
       dw    dummy_isr           ; CAN Wakeup Vector
       dw    dummy_isr           ; SPI Transmit Vector
       dw    dummy_isr           ; SPI Receive Vector
       dw    dummy_isr           ; TIMB Overflow Vector
       dw    dummy_isr           ; TIMB Channel 1 Vector
       dw    dummy_isr           ; TIMB Channel 0 Vector
       dw    dummy_isr           ; TIMA Overflow Vector
       dw    dummy_isr           ; TIMA Channel 3 Vector
       dw    dummy_isr           ; TIMA Channel 2 Vector
```

```
        dw    dummy_isr          ; TIMA Channel 1 Vector
        dw    dummy_isr          ; TIMA Channel 0 Vector
        dw    dummy_isr          ; TIM Vector
        dw    dummy_isr          ; PLL Vector
        dw    dummy_isr          ; ~IRQ1
        dw    dummy_isr          ; SWI Vector
        dw    main_init          ; Reset Vector
;
; You may use this code freely as long as this copyright notice
; and website address is included. Visit us at www.pemicro.com
```

So how is the code laid out in order to handle this? The beginning of the code shows symbol defines for addresses within memory for RAM, flash, and interrupt vectors. The CAN address is specified here as well. Also included are the register file definitions, which give the addresses of peripheral registers on the microcontroller. Following this comes the variable declarations for pointers, counters, and buffers for the received data.

In the flash ROM, the message descriptions are given, which indicate the destination address, message priority, message length, and the actual data. Note that the number of data bytes can be variable sizes from 0-8. The information needed in these descriptions can be found in the CAN specification and the 68HC908AZ60A datasheet.

After the message descriptions comes the actual code, where there are some procedures for handling different tasks. The first is the initialization procedure, which sets up the registers in the CAN peripheral for the intended operation. The CAN peripheral is also enabled at the end of the routine. After this comes the start of the application in the MAIN_INIT procedure, where we call the initialization routines, initialize the pointers and counters, and wait for an interrupt to occur. There is one interrupt service routines for the CAN peripheral in this code which is triggered when the peripheral receives CAN messages specifically intended for it. Note that this application is totally interrupt driven, where the code is driven by any events that take place through the CAN peripheral.

The end of the code shows the interrupt vectors for the microcontroller, where the CAN interrupts and the RESET vector are included. All other vectors are pointed to a dummy interrupt service routine that simply just returns from the interrupt.

Once the mentioned above assembly code is compiled as a part of the Freescale project, one can step through in the Full Chip Simulation mode. The execution will automatically start at the reset location within the code. We are now ready to start debugging the code.

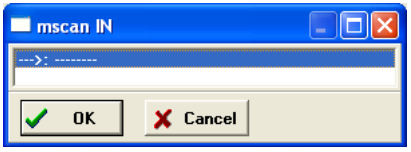
First, we need to step through the beginning of the code in the simulator in order to allow the code to initialize the microcontroller and the CAN peripheral and get ready to start receiving CAN packets from the network. So, the user needs to step through the beginning of the code until they get to the main loop with the t command.

At this point we are ready for CAN packets to come into the device. There are three specific commands in the simulator for testing the CAN peripheral: CANIN, CANOUT, and CANCLR (see detailed MSCAN commands description below). The CANIN

command allows a user to specify messages coming into the CAN peripheral. The CANOUT command allows a user to see the packets that were sent out from the CAN peripheral. The CANCLR command allows the user to clear all input and output buffers of CAN packets.

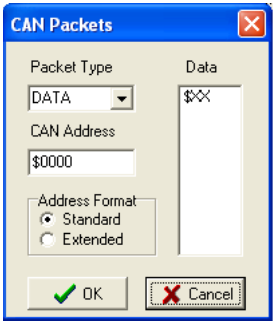
Now we can use the CANIN command to specify CAN packets that come from the network to our device. When a user types this command in the simulator, the CAN IN window appears as shown in Figure 11.27 on page 283. There are no CAN packets specified yet in the window, so we will enter some into it.

Figure 11.27 CAN IN Display



Input packets can be added by double-clicking in the window where the packet is to go or by selecting the row for the packet and clicking the OK button. At this point a new window comes up as shown in Figure 11.28 on page 283. It is in this window that the user can specify the parameters of the message. The “Packet Type” pull down menu allows the user to select the type of CAN packet, where it could be a DATA, REMOTE, ERROR, or OVERLOAD packet. The CAN address for the input can be entered in the CAN Address edit box. The address format can be set to be standard (11 bits) or extended (29 bits). Note that if a ERROR or OVERLOAD packet is selected, then the DATA field and CAN Address fields are not utilized and are grayed out. For DATA and REMOTE packets, the DATA field must be filled with the appropriate data. Let’s show how to fill these fields for a data packet coming into the device from the network.

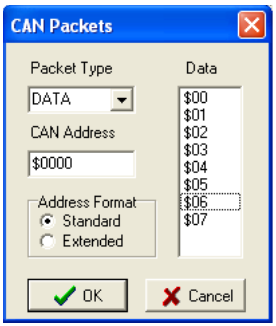
Figure 11.28 CAN Packet Dialog Box



As an example, let us assume that another node in the network wanted to send a packet containing 8 bytes (0,1,2,3,4,5,6,7) to our device at address $01234567. In this case we want to set the packet type to DATA, the CAN address to $01234567, the address format

to Extended, and the data field for the given 8 bytes. This is shown in Figure 11.29 on page 284 through the CANIN command.

Figure 11.29 CAN Example Input Packet



Once the DATA packet information is entered, the CANIN window should look as shown in Figure 11.30 on page 284. The user can then set a breakpoint in the interrupt service routine that handles CAN packets and execute the code by typing “GO” in the simulator. The user should now be at the CAN ISR. By entering the “t” command and single stepping through the code, the user will exit the ISR and return to the main loop. The user can then see the resulting data in the memory window pointed to by the buffptr variable, which is the 0,1,2,3,4,5,6,7 placed in the data packet. The memory window is shown in Figure 11.31 on page 284.

Figure 11.30 CAN IN Display After Data Packet

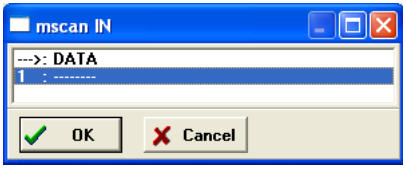
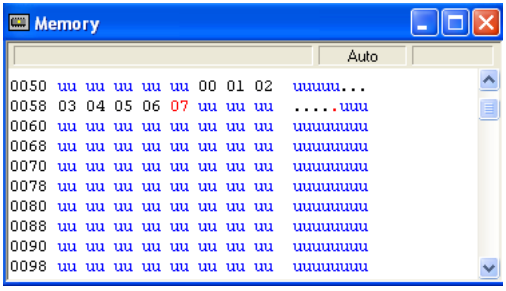


Figure 11.31 Memory Window Showing Received Data


Memory
Auto
0050 uu uu uu uu uu 00 01 02 uuuuu...
0058 03 04 05 06 07 uu uu uu .....uuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu
0088 uu uu uu uu uu uu uu uu uuuuuuuu
0090 uu uu uu uu uu uu uu uu uuuuuuuu
0098 uu uu uu uu uu uu uu uu uuuuuuuu

Now, for the output packets that were setup in the initialization routine, we can see that the code sends them out from the simulator by using the CANOUT command. The user can type the command GO in the simulator and let the simulation run for a while. Hit the Enter key to break the execution. Now type the CANOUT command. Figure 11.32 on page 285 shows the result, where all three output packets are seen. Select the first packet by double clicking on it. Figure 11.33 on page 285 shows the first CAN packet sent out. Note that the first packet sent out was the packet with the highest priority. You can now select the other packets in the CAN OUT window to open them up and see which data packet they were.

Figure 11.32 CAN OUT Display

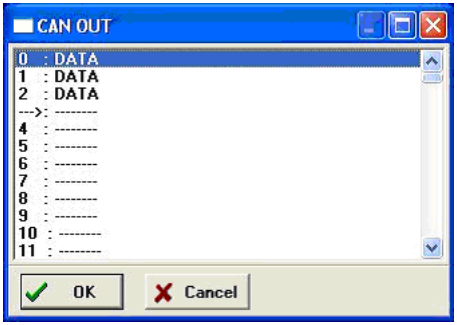
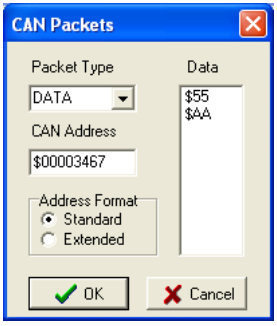


Figure 11.33 CAN Example Output Packet



It is possible that a user would not want to have to enter all of these simulator commands manually. Instead for automated testing of their code, they could always use macro files. A macro file permits a user to setup a sequence of simulator commands to be run one after the other within the simulator. So, the CANIN command can include parameters that specify the type of packet, CAN address, and data. See the CANIN command in the help file for more details.

FCSMSCAN Commands

CANCLR Command

The CANCLR command can be used to flush the input and output buffers for CAN simulation. This will reset the buffers and clear out all values. Notice that if the CAN is currently shifting a value, this command will not prevent the CAN from finishing the transfer. See CANIN Command and CANOUT Command for accessing the input and output buffers of the CAN interface.

Syntax:

```
>CANCLR
```

Example:

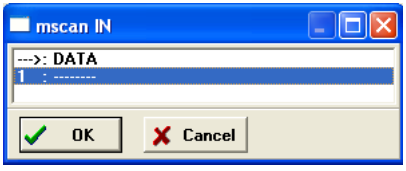
```
>CANCLR
```

Clear input and output buffers for CAN simulation

CANIN Command

The CANIN command allows the user to input data into the CAN. If a data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if not parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open.

Figure 11.34 MSCAN_IN Buffer



An arrow points to the value that will be used next as input to the CAN. The maximum number of input packets is 256.

Syntax:

```
>CANIN [<n>]
```

Where <n> is the value to be entered into the next location in the input buffer.

Example:

```
>CANIN $55
```

Set the next input CAN value to $55

```
>CAN
```

Pull up the data window with all the input packets.

CANOUT Command

The CANOUT command displays the output of the buffer from the CANOUT. A window is opened that shows all the data that the CAN has shifted out. An arrow is used to point to the last output value transmitted. The maximum number of output packets that the buffer holds is 256 bytes.

Syntax:

```
CANOUT
```

Example:

```
>CANOUT
```

View data from the output buffer for CAN simulation

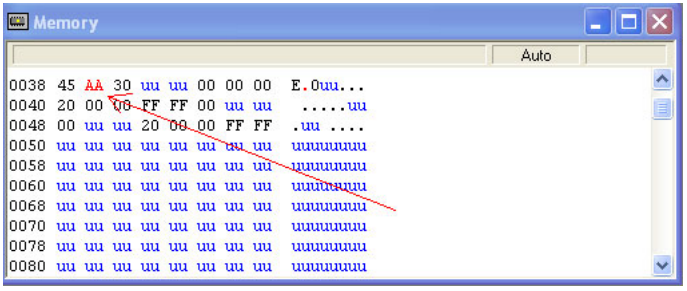
FCSProgrammable Timer Interrupt Module

In Full Chip Simulation Mode, this module will simulate all functionality of the Programmable Timer Interrupt (PIT) module, including:

- Programmable PIT clock input
- Free running or modulo up count operation
- Flag polling
- Interrupt enabled mode of operation

Once the PIT Status and Control register properly configures the operation of the module, the PIT Counter will start incrementing. If modulo up count operation is enabled, the PIT overflow flag can be observed in the PIT Status and Control register in the Memory Window.

Figure 11.35 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

If the PIT interrupt is enabled, the FCS will jump to an appropriate subroutine as long as the PIT interrupt vector is properly defined.

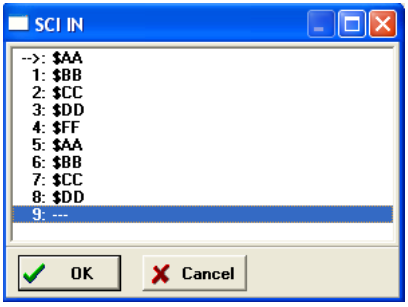
FCSSerial Communications Interface Module

In Full Chip Simulation Mode, this module will simulate all functionality of the Serial Peripheral Interface (SPI) module including:

- flag polling
- interrupt enabled mode
- 8- or 9-bit length data codes
- odd and even parity modes
- transmission and reception of external data

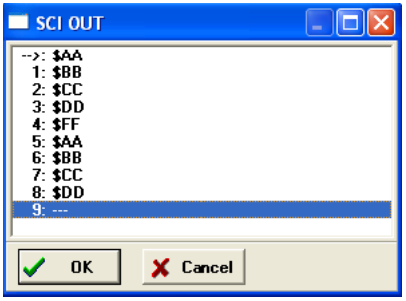
Full Chip Simulation (FCS) mode uses the buffered input/output structure to simulate SCI inputs. The user can queue up to 256 data values into the input buffer. The output buffer of the SCI module can also hold 256 output values. To queue the SCI Input Data, one should use the SCDI <n> command in the command prompt. If <n> (the data parameter) is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is provided, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the SCI. The data from the SCI input buffer is written to the SCI data register once the SCI module has been turned on and is properly configured for receiving data from an external serial device. Once the simulation of the data transmission is over, the arrow moves to the next value in the SCI IN Buffer.

Figure 11.36 SCI IN Buffer Display



SCI Data Output Buffer simulation allows the user to gain access to the past 256 SCI data values transmitted out of the module. To bring up the SCI OUT buffer dialog, the SCDO command should be used.

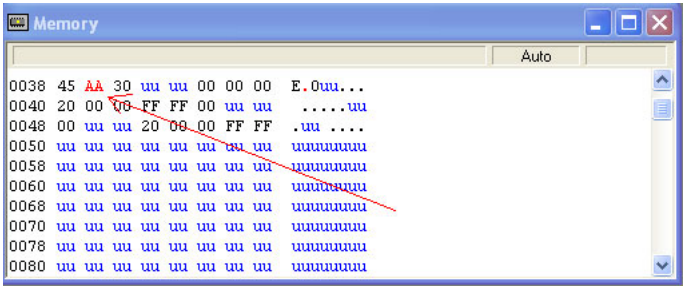
Figure 11.37 SCI OUT Buffer Display



At any point, the SCCLR command can flush the input and output SCI buffers.

After the SCI simulated input is received, the first queued value is passed from the data buffer into the SCI data register. It can be observed in the memory window by displaying the memory location corresponding to the SCI data register.

Figure 11.38 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

The user can also observe different SCI flags in the Memory window. If the module is run in Flag Polling mode, the user should poll the flag corresponding to the expected SCI event. If the SCI interrupts are enabled, the FCS will jump to an appropriate subroutine as long as the SCI interrupt vectors are properly defined.

For more information on how to configure SCI module for desired operation, refer to the Freescale user manual corresponding to the Freescale microprocessor that you are using..

FCSSCI Commands

SCCLR Command

The SCCLR command can be used to flush the input and output buffers for SCI simulation. This will reset the buffers and clear out all values. Notice that if the SCI is in the process of shifting a value, this command will not prevent the SCI from finishing the

transfer. See SCDI command and SCDO command for accessing the input and output buffers of the SCI interface.

Syntax:

```
>SCCLR
```

Example

```
>SCCLR
```

Clear input and output buffer for SCI simulation

SCDI Command

The SCDI command allows the user to input data into the SCI. If a data parameter is given, the value is placed into the next slot in the SCI input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the SCI. The maximum number of input values is 256 bytes.

Syntax:

```
>SCDI [<n>]
```

Where:

><n> The value to be entered into the next location in the input buffer

Example:

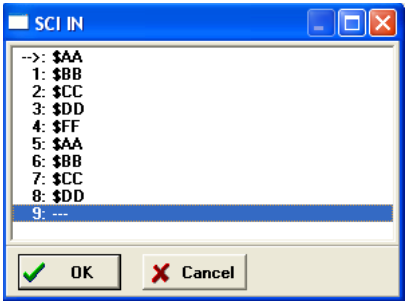
```
>SCDI $55
```

Set the next input value to the SCI to $55

```
>SCDI
```

Pull up the data window with all the input values.

Figure 11.39 SCI IN Buffer Display



SCDO Command

The SCDO command displays the output buffer from the SCI. A window is opened that shows all the data that the SCI has shifted out. An arrow is used to point to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

Syntax:

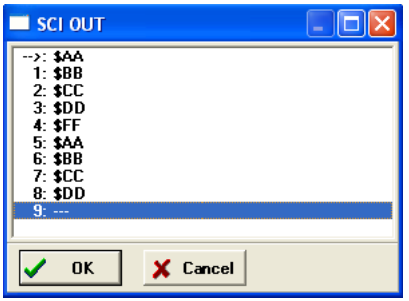
```
>SCDO
```

Example:

```
>SCDO
```

View data from the output buffer for the SCI simulation.

Figure 11.40 SCI OUT Buffer Display



FCSSlave LIN Interface Controller Module

In Full Chip Simulation Mode, this module will simulate all functionality of the Slave LIN Interface Controller (SLIC) Module, including:

- Flag polling
- Interrupt enabled mode
- Transmission and reception of external data
- Check sum generation and verification
- Different message lengths data modes

The Full Chip Simulation mode uses a buffered structure to simulate SLIC inputs and outputs. The user can queue up to 256 data bytes into the input buffer. The output buffer of the SLIC module can also hold 256 output bytes. To queue the SLIC Input bytes, one should use the SLCIN command in the command prompt. For a more detailed description of the command, please refer to the SLIC Commands section. The SLIC command brings up a window, which displays a list of queued in input data. Different SLIC packets can be

entered while the window is open. An arrow points to the byte that will be used next as input to the SLIC. Once the SLIC module is turned on and properly configured for receiving data from an external SLIC device, the data from the SLIC input buffer is written to the SLIC module identifier or data registers. After the simulation of the data transmission is complete, the arrow moves to the next value in the SLIC IN Buffer.

Figure 11.41 SLIC IN Buffer Display

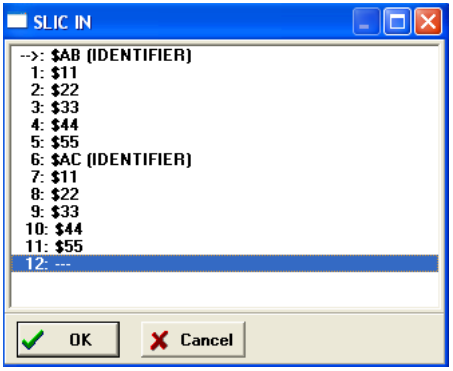
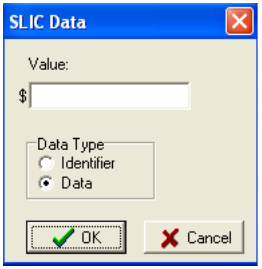
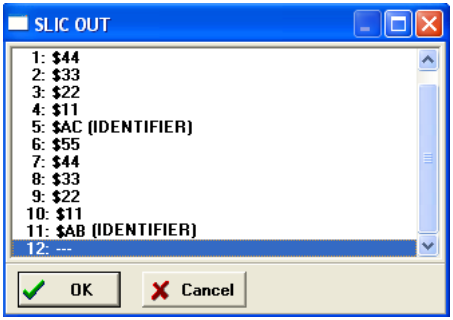


Figure 11.42 SLIC Input Data Configuration Dialog Box



The SLIC data output buffer simulation allows the user to gain access to the past 256 SLIC data bytes transmitted out of the module. To bring up the SLIC OUT buffer dialog, the SLC0UT command should be used.

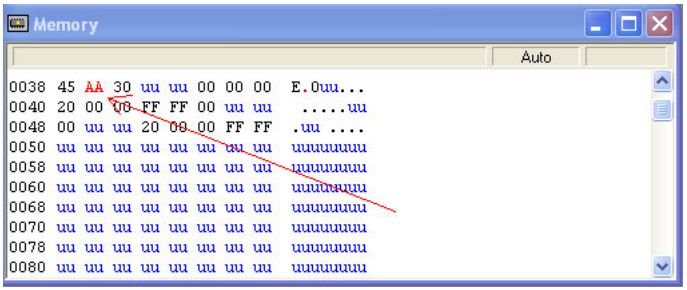
Figure 11.43 SLIC OUT Buffer Display



At any point, the SLCCLR command can flush the input and output SLIC buffers.

After the simulated SLIC input is received, the first queued-in packet is passed from the data buffer into the corresponding SLIC module registers. It can be observed in the Memory Window by displaying the appropriate register location there.

Figure 11.44 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

The user can also observe different SLIC flags in the Memory window. If the module is run in Flag Polling mode, the user should poll the flag corresponding to the expected SLIC event. If the SLIC interrupts are enabled, the FCS will jump to an appropriate subroutine, as long as the SLIC interrupt vectors are properly defined. Please note that the SLIC State Vector Register reflects the specific SLIC interrupt that was triggered. CPU overhead for servicing different LIN interrupts can be significantly decreased by monitoring the state of this register from within the interrupt subroutine.

For more information on how to configure SLIC module for desired operation, please refer to the Freescale manual corresponding to the microprocessor that you are using.

FCSSLIC Commands

SLCCLR Command

The SLCLR command can be used to flush the input and output buffers for SLIC simulation. This will reset the buffers and clear out all packets. Notice that if the SLIC is currently shifting a value, this command will not prevent the SLIC from finishing the transfer. See SLCDI command and SLCDOUT command for accessing the input and output buffers of the SLIC interface.

Syntax:

```
>SLCCLR
```

Example

```
>SLCCLR
```

Clear input and output buffer for SLC simulation

SLCDI Command

The SLCDI command is used to simulate SLIC input packets. The first input should always be of type “identifier.” Subsequent bytes may either be “data” or “identifier.” After the SLIC Data IN buffer is filled with some data, the SLIC simulation module will begin reception of the data packet as soon as it is properly configured and turned on within the user’s firmware. In Byte Transfer Mode, the designation of a byte as either “data” or “identifier” will have no effect. The SLIC data input buffer can store up to 256 bytes.

Syntax:

SLCDI

Example

SLCDI

Bring up the SLIC IN buffer.

Figure 11.45 SLIC Data Input Configuration Dialog Box

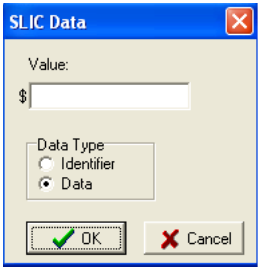
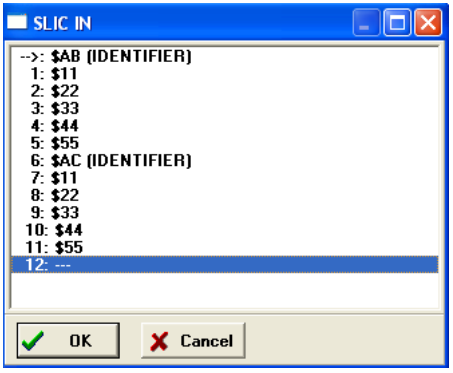


Figure 11.46 SLIC Input Data Buffer



SLCOUT Command

The SLCOUT command displays the output buffer from the SLIC. A window is opened that shows all the data that the SLIC has shifted out, in either Byte Transfer or SLIC Mode. An arrow is used to point to the last output packet transmitted. The maximum number of output packets that the buffer holds is 256 bytes.

Syntax:

```
>SLCOUT
```

Example:

```
>SLCOUT
```

View packets from the output buffer for SLIC simulation.

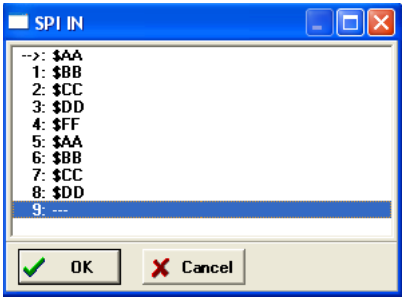
FCSSerial Peripheral Interface Module

In Full Chip Simulation Mode, this module will simulate all functionality of the Serial Peripheral Interface (SPI) module including:

- Flag polling
- Interrupt enabled mode
- Master and slave modes
- Slave input clock
- Transmission and reception of external data

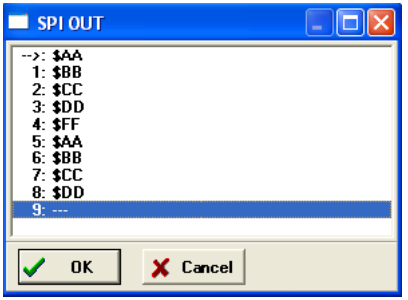
The Full Chip Simulation (FCS) mode uses the buffered input/output structure to simulate SPI inputs. The user can queue up to 256 data values into the input buffer. The output buffer of the SPI module can also hold 256 output values. To queue the SPI Input Data, one should use the SPDI <n> command at the command prompt. If <n> (the data parameter) is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is provided, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the SPI. The data from the SPI input buffer is written to the SPI data register once the SPI module has been turned on and is properly configured for receiving data from an external serial device. Once the simulation of the data transmission is over, the arrow moves to the next value in the SPI IN Buffer.

Figure 11.47 SPI IN Buffer Display



SPI data output buffer simulation allows the user to gain access to the past 256 SPI data values transmitted out of the module. To bring up the SPI OUT buffer dialog, the SPDO command should be used.

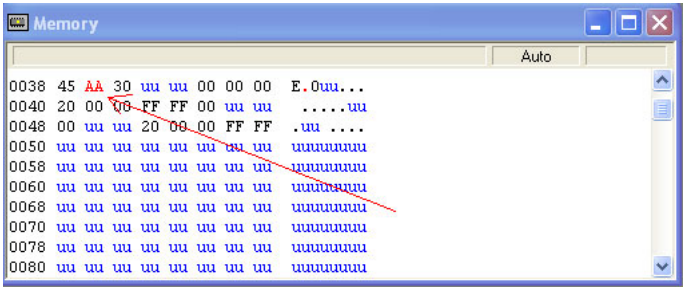
Figure 11.48 SPI OUT Buffer Display



At any point, SPCLR command can flush the input as well as output SPI buffers.

After the SPI simulated input is received, the first queued value is passed from the data buffer into the SPI data register. It can be observed in the Memory Window by displaying the memory location corresponding to the SPI data register.

Figure 11.49 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

The user can also observe different SPI flags, in the Memory window. If the module is run in the Flag Polling mode, the user should poll the flag corresponding to the expected SPI event. If the SPI interrupts are enabled, the FCS will jump to an appropriate subroutine as long as the SPI channel interrupt vectors are properly defined.

To simulate the frequency of the SPI slave input clock, the SPFREQ <n> command is used. If the SPI is configured for slave mode, this command allows the user to enter the number of cycles <n> in the period of the input clock. If the SPFREQ command is not used, then clocking is set by the SPI control register.

For more information on how to configure the SPI module for desired operation, refer to the Freescale manual corresponding to the microprocessor that you are using.

FCSSPI Commands

SPCLR Command

The SPCLR command can be used to flush the input and output buffers for SPI simulation. This will reset the buffers and clear out all values. Notice that if the SPI is currently shifting a value, this command will not prevent the SPI from finishing the transfer. See SPDI command and SPDO command for accessing the input and output buffers of the SPI interface.

Syntax:

```
>SPCLR
```

Example

```
>SPCLR
```

Clear input and output buffer for SPI simulation

SPDI Command

The SPDI command allows the user to input data into the SPI. If a data parameter is given, the value is placed into the next slot in the SPI input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the SPI. The maximum number of input values is 256 bytes.

Syntax:

```
>SPDI [<n>]
```

Where:

<n> The value to be entered into the next location in the input buffer

Example:

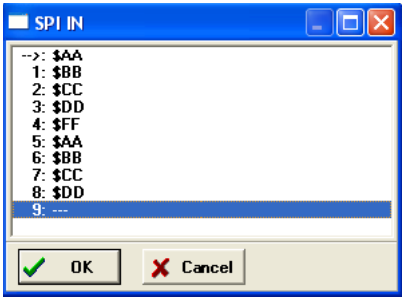
```
>SPDI $55
```

Set the next input value to the SPI to $55

```
>SPDI
```

Pull up the data window with all the input values.

Figure 11.50 SPI IN Buffer Display



SPDO Command

The SPDO command displays the output buffer from the SPI. A window is opened that shows all the data that the SPI has shifted out. An arrow is used to point to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

Syntax:

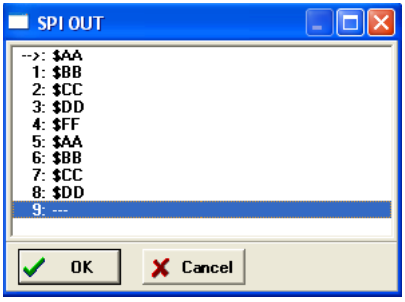
```
>SPDO
```

Example:

```
>SPDO
```

View data from the output buffer for the SPI simulation.

Figure 11.51 SPI OUT Buffer Display



SPFREQ Command

The SPFREQ command lets the user set the frequency of the SPI slave input clock. If the SPI is configured for the slave mode, this command allows the user to enter the number of cycles <n> per one input clock period. If no value is given, a popup window will appear

and the user will be prompted for a value. If this command is not used, then the clocking is assumed to be set by the SPI control register.

Syntax:

```
>SPFREQ [<n>]
```

Where:

<n> The number of cycles for the period of the input clock.

Example:

```
>SPFREQ 8
```

Set the period of the input slave clock to 8 cycles (total shift = 8*8 cycles per bit = 64 cycles)

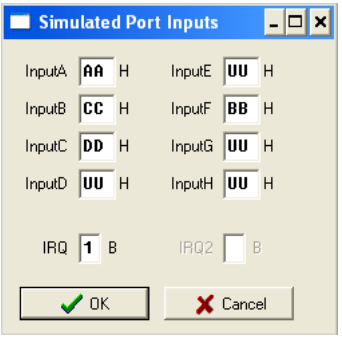
FCSTimer Interface Module

In Full Chip Simulation Mode, this module will simulate all functionality of the Timer Interface module, including:

- Input capture/output compare
- Pulse width modulation
- Internal or external clock input
- Free running or modulo up count operation
- Flag polling
- Interrupt enabled mode of operation.

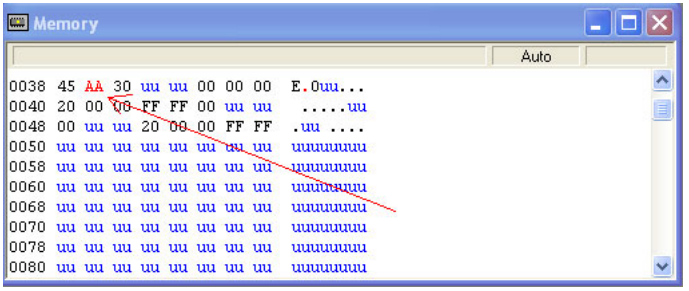
The Full Chip Simulation (FCS) mode uses the simulated port inputs to trigger the input capture on a given timer channel. To define an input state of the specific port one should use the INPUT<x> <n> command in the Command line window. The <x> represents the corresponding I/O port, while <n> stands for the input value that should be written to this port. At the same time, the INPUTS command can be used to display the Simulated Port Inputs for all general IO ports. It displays the current simulated values to all applicable input ports. Please see the documentation for the command INPUT<x>/INPUTS for more information about the various forms of this command.

Figure 11.52 Simulated Port Inputs Dialog Box



Using the Simulated Port Inputs dialog box, the input value to any I/O port can be reconfigured. Depending on whether the input capture is set for rising/falling edge, to trigger the event, the inputs to the port should first be set high/low and then be inverted to an opposite value. Once the Input Capture event takes place, the CHxF can be observed in the Channel Status and Control register in the Memory window.

Figure 11.53 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

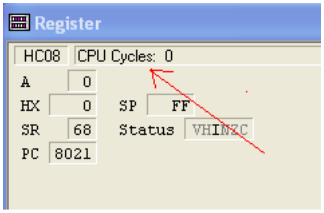
If the Timer module is configured for an Output Compare event, once the event takes place the same CHxF Flag can be observed via the Memory window. If the timer channel interrupt is enabled, the FCS will jump to an appropriate subroutine as long as the Timer channel interrupt vector is properly defined. To observe the Timer Overflow Flag (TOF) flag going up as a result of the corresponding CPU event, situate your Memory window on the memory location of the Timer Status and Control register.

To observe the Pulse Width Modulation (PWM) operation, properly configure the Timer to operate in the Modulo up count mode, choose the toggle-on-overflow or clear/set output on compare events to create a desired duty cycle wave. Once a PWM event takes place, pin toggle/clear/set behavior corresponding to the Timer configuration can be observed in the Memory window displaying the IO port associated with a given timer channel.

To observe the accuracy of the Timer module operation, the user can observe the number of CPU cycles that it takes for the event to occur. The cycle counter is only incremented as

the user steps through the code. To determine the exact amount of cycles over which the event occurs, one can either observe the cycle display in the Register window or use the built in simulation commands. To display the current number of cycles in the Command window, the CYCLES command should be used. To change the number of cycles in the cycle counter, use CYCLES <n>, where <n> is the new cycle value. If the number of cycles that an event should take has already been pre-calculated, one can use CYCLE 00 to reset the number of cycles and GOTOCYCLE <n> to run through the code until the place where expected event should take place.

Figure 11.54 Register Window With Cycles Display



FCSTimer Interface Module Commands

CYCLES Command

The CYCLES command changes the value of the cycles counter. The cycles counter counts the number of the processor cycles that have passed during execution. The Cycles Window shows the cycle counter. The cycle count can be useful for timing procedures.

Syntax:

```
>CYCLES <n>
```

Where:

<n> Integer value for the cycles counter

Examples:

```
>CYCLES 0
```

Reset cycles counter

```
>CYCLES 1000
```

Set cycle counter to 1000.

GOTOCYCLE Command

The GOTOCYCLE command executes the program in the simulator beginning at the address in the program counter (PC). Execution continues until the cycle counter is equal to or greater than the specified value, until a key or the Stop button on the toolbar is pressed, until it reaches a break point, or until an error occurs.

Syntax:

```
GOTOCYCLE <n>
```

Where:

<n> Cycle-counter value at which the execution stops

Example:

```
>GOTOCYCLE 100
```

Execute the program until the cycle counter equal 100.

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax:

```
INPUT<x>  <n>
```

Where:

<x> is the letter representing corresponding port

<n> Eight-bit simulated value for port <x>

Example:

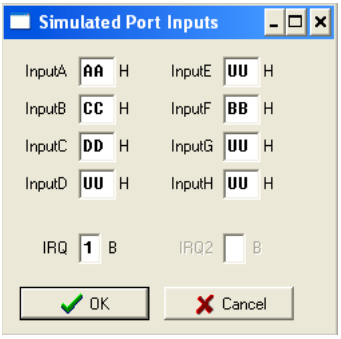
```
>INPUTA AA
```

Simulate the input AA on port A.

INPUTS Command

In Full Chip Simulation and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 11.55 on page 304. Use this box to specify the input states of port pins and IRQ.

Figure 11.55 Simulated Port Inputs Dialog Box



When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax:

```
>INPUTS
```

Example:

```
>INPUTS
```

Show I/O port input values.

FCSUniversal Serial Bus (USB) Module

Some of the microcontrollers in the 68HC08 family contain USB compliant peripheral devices. These can be low-speed or high-speed USB slave devices. This means that all USB transfers are initiated by a host (i.e. a personal computer) and that the microcontroller needs to be setup to respond with the appropriate acknowledgement messages. According to the USB specification, there are a series of messages that go back and forth between the host and the device in order to setup and describe the channel for data transfer. All of these messages can be set up through the USB I/O commands built into the full chip simulator.

This full chip simulator mode will simulate all functionality of the Universal Serial Bus (USB) module including

- Flag polling
- Interrupt enabled mode
- Transmission and reception of external data
- Endpoint 0/1/2 modes of operation
- USB reset functionality
- STALL, NAK and ACK handshakes

This section goes through an example of assembly code, showing how the simulator can be used to test out code for driving the USB peripheral.

Listing 11.2 on page 306 shows an assembly program that exercises the USB full speed peripheral on the 68HC908JW32. The source assembly file can be download from P&E Microcomputer website at www.pemicro.com. With minor modifications this assembly file can be added and compiled as a part of your Freescale project. It sets the microcontroller up as a simple USB human interface device (HID) and uses interrupts from the USB peripheral to service requests from the host. This code can be assembled and then loaded into the ICS08 simulator for execution, after which simulator commands can be used to emulate the USB packets that come from the host. Let us first break down what is being done in the example program.

Listing 11.2 68HC908JW32 USB HID Demonstration Application

```
; 68HC908JW32 USB HID Demonstration Application
; (C)opyright P&E Microcomputer Systems, 2005
;
; You may use this code freely as long as this copyright notice
; and website address is included. Visit us at www.pemicro.com

; This application is meant to demonstrate a framework for an
application
; running on the 68HC908JW32. It demonstrates a simple HID interface
for a
; USB device and uses interrupts from the USB peripheral. The HID
interface
; is supported through standard API calls in Windows XP and Windows
2000.
; For more information on USB, please visit the USB developers website
at
; www.usb.org and download the USB specification revision 2.0 and the
HID
; device class specification version 1.1 for more details.

RAMStart        equ    $0060               ; start of RAM for JW32
RomStart        equ    $7000               ; start of Flash for JW32
VectorStart     equ    $FFEE               ; start of Vectors for JW32
EP1BuffStart    equ    $1000             ; start of buffer for endpoint
1 of the USB
EP2BuffStart    equ    $1010             ; start of buffer for endpoint
2 of the USB

$Include 'JW32regs.inc'

    org RamStart
Setup_Packet     ds      8      ; array for SETUP packet
bmReqType       equ    Setup_packet        ; Characteristic of Request
bRequest        equ    {Setup_packet+1}    ; Request Code
wValueL         equ    {Setup_packet+2}    ; Low byte Value Field
wValueH         equ    {Setup_packet+3}    ; High byte Value Field
wIndexL         equ    {Setup_packet+4}    ; Low byte Index Field
wIndexH         equ    {Setup_packet+5}    ; High byte Index Field
wLengthL        equ    {Setup_packet+6}    ; Low byte Length Field
wLengthH        equ    {Setup_packet+7}    ; High byte Length Field

GET_DESC      equ     6          ; Standard Request code for
GET_DESCRIPTOR
SYNC_FRAME    equ     12t        ; Standard Request code for SYNC_FRAME

control       ds      1          ; type of transfer in progress
```

```
descptr       ds      2         ; pointer to descriptor being sent
descendptr    ds      2         ; end pointer to descriptor being sent
buffptr       ds      2         ; index into the USB data buffer
tptr          ds      2         ; temporary pointer for storage
EP2data       ds      8         ; endpoint 2 output received data

    org RomStart

* The following descriptors give the information to the PC what type of
* USB device this is and what its capabilities are. They are retrieved
* during the configuration phase.

* Note that the Vendor and Product IDs specified in this demo are
invalid
* USB IDs and are given for demonstration purposes only!

* Device Descriptor
Dev_Desc:
       db  {DDesc_End-Dev_Desc} ; Descriptor Length
       db  $01                  ; Descriptor Type (Device)
       db  $00,$02              ; USB specification Release (2.00)
       db  $00                  ; Class Code
       db  $00                  ; Subclass Code
       db  $00                  ; Protocol Code
       db  $08                 ; Maximum Packet Size for EP0 (8 bytes)
       db  $00,$00              ; Vendor ID=none
       db  $00,$00              ; Product ID=none
       db  $01,$00              ; Device Release Number (1.00)
      db  $01                ; Index to Manufacturer String Descriptor
       db  $02                  ; Index to Product String Descriptor
       db  $00                 ; Index to Device Serial Number String
Descriptor
       db  $01                 ; Number of possible configurations (1)
DDesc_End:

* Configuration Descriptor
Con_Desc:
       db  {CDesc_End-Con_Desc} ; Descriptor Length
       db  $02                  ; Descriptor Type (Configuration)
       db  {E2Desc_End-Con_Desc},$00 ; Total data length (Config-
Interface-EP)
       db  $01                  ; Interfaces supported
       db  $01                  ; Configuration Value
       db  $00                  ; Index to String Descriptor
       db  $C0                  ; Self powered
       db  $00                  ; Maximum power consumption=0mA (not
applicable)
CDesc_End:
```

```
* Interface Descriptor
Int_Desc:
       db  {IDesc_End-Int_Desc} ; Descriptor Length
       db  $04                  ; Descriptor Type (Interface)
       db  $00                  ; Number of Interface
       db  $00                  ; No alternate setting
       db  $02                  ; Number of endpoints
       db  $03                  ; Class Code (HID)
       db  $00                  ; Subclass Code
       db  $00                  ; Protocol Code
       db  $00                  ; Index to String Descriptor
IDesc_End:

* HID Descriptor
HID_Desc:
       db  {HDesc_End-HID_Desc} ; Descriptor Length
       db  $21                  ; Descriptor Type (HID)
       db  $00,$01              ; HID Class Release (1.00)
       db  $00                  ; Country Code=$00
       db  $01                  ; number of HID class descriptors
       db  $22                  ; Class Descriptor Type (REPORT)
       db  {RDesc_End-Rep_Desc},$00 ; length of report descriptor
HDesc_End:

* Endpoint 1 Descriptor
Endp1_Desc:
       db  {E1Desc_End-Endp1_Desc} ; Descriptor Length
       db  $05                  ; Descriptor Type (Endpoint)
       db  $81                  ; Endpoint Number and Direction (#1,IN)
       db  $03                  ; Endpoint Attribute (Interrupt)
       db  $01,$00              ; Maximum Packet Size for EP1 (1 byte)
       db  $FF                  ; Polling Interval=255[ms]
E1Desc_End:

* Endpoint 2 Descriptor
Endp2_Desc:
       db  {E2Desc_End-Endp2_Desc} ; Descriptor Length
       db  $05                  ; Descriptor Type (Endpoint)
       db  $02                  ; Endpoint Number and Direction (#2,OUT)
       db  $02                  ; Endpoint Attribute (Bulk)
       db  $08,$00              ; Maximum Packet Size for EP2 (8 bytes)
       db  $FF                  ; Polling Interval=255[ms]
E2Desc_End:

* Report Descriptor
Rep_Desc:
       db  $06,$00,$FF          ; Usage Page (vendor defined)
```

```
       db  $09,$01               ; Usage (vendor defined)

       db  $A1,$01               ; Collection (Application)
       db  $09,$02               ; Usage (vendor defined)

; Input report
       db  $09,$03               ; Usage (vendor defined)
       db  $15,$00               ; Logical Minimum ($00)
       db  $26,$FF,$00           ; Logical Maximum ($FF)
       db  $75,$08               ; Report Size (8 bits)
       db  $95,$01               ; Report Count (1 field)
       db  $81,$02               ; Input (Data, Variable, Absolute)

       db  $C0                   ; End Collection
RDesc_End:

*************************************************************
* Init_USB - Disables receive and transmit for all endpoints. *
*            The USB state is set to powered, where the part  *
*            is waiting for an USB reset and for it to be     *
*            addressed and configured.                        *
*************************************************************
Init_USB:
       clr   control             ; waiting for control packets
       mov   #$00,USBSR          ; Reset all USB flags
       mov   #$2d,USIMR         ; Enable interrupts for config change,
setup, reset, and suspend events
      mov   #$d0,UEP1CSR       ; enable EP1 as interrupt, IN direction,
buffer size of 8
       mov   #$80,UEP2CSR        ; enable EP2 as bulk, OUT direction,
buffer size of 8
       mov   #$20,UEP12BPR       ; set base address pointer for EP1 to
$1000, EP2 to $1010
       mov   #$00,UINTFCR       ; set interface number for EP1,EP2 to 0
       mov   #$4e,USBCR          ; turn on USB clocking, enable
interrupts for endpoint 0,1,2 transfers
       mov   #$ce,USBCR          ; enable USB module
       rts


*************************************************************
* FORCE_STALL - A packet is received in the control stage that *
*               is not supported. So the device stalls until   *
*               new SETUP packet arrives at endpoint 0.        *
*************************************************************
FORCE_STALL:
       lda   USIMR               ; request not handled
       ora   #40                 ; set EP0_STALL
```

```
       sta   USIMR              ; new SETUP packet clears STALL
       rts                      ;  bits automatically

*****************************************************************
* GETDESC_PROC - This procedure handles the standard request   *
*                to get the device's descriptors.               *
*****************************************************************
GETDESC_PROC:
       ldhx  #0                 ; clear H:X pointer
       lda   wValueH            ; check which descriptor is wanted
       cbeqa #$1,GETDEVDESC     ; is it device descriptor?
       cbeqa #$2,GETCONDESC     ; is it configuration descriptor?
       cbeqa #$21,GETHIDDESC    ; is it for HID descriptor?
       cbeqa #$22,GETREPDESC    ; is it for Report descriptor?
       jmp   GETDESC_STALL      ; else go stall

GETDEVDESC:
       lda   Dev_Desc,x         ; take device descriptor information
       sta   UE0D0,x            ; store in USB endpoint 0 data buffer
       incx
       cpx   #8                 ; all descriptors more than 8 bytes
       bne   GETDEVDESC
       ldhx  #DDesc_End         ; store end location of the descriptor
       sthx  descendptr
       ldhx  #Dev_Desc          ; store pointer to next byte in
descriptor
       bra   GETDESC_END

GETCONDESC:
       lda   Con_Desc,x         ; take configuration descriptor
information
       sta   UE0D0,x            ; store in USB endpoint 0 data buffer
       incx
       cpx   #8                 ; all descriptors more than 8 bytes
       bne   GETCONDESC
       ldhx  #E2Desc_End        ; store end location of the descriptor
       sthx  descendptr
       ldhx  #Con_Desc          ; store pointer to next byte in
descriptor
       bra   GETDESC_END

GETHIDDESC:
       lda   HID_Desc,x         ; take HID descriptor information
       sta   UE0D0,x            ; store in USB endpoint 0 data buffer
       incx
       cpx   #8                 ; all descriptors more than 8 bytes
       bne   GETHIDDESC
       ldhx  #HDesc_End         ; store end location of the descriptor
```

```
       sthx  descendptr
       ldhx  #HID_Desc          ; store pointer to next byte in
descriptor
       bra   GETDESC_END

GETREPDESC:
       lda   REP_Desc,x         ; take report descriptor information
       sta   UE0D0,x            ; store in USB endpoint 0 data buffer
       incx
       cpx   #8                 ; all descriptors more than 8 bytes
       bne   GETREPDESC
       ldhx  #RDesc_End         ; store end location of the descriptor
       sthx  descendptr
       ldhx  #REP_Desc          ; store pointer to next byte in
descriptor

GETDESC_END:
       sthx  descptr
       lda   descptr+1          ; add length specified in setup packet
       add   wLengthL           ;  and store in tptr
       sta   tptr+1
       lda   descptr
       adc   wLengthH
       sta   tptr
       ldhx  tptr               ; is calculated pointer >= end of
descriptor?
       cphx  descendptr         ; if so then end_pointer = end of
descriptor
      bge   GETDESC_END2       ; else, end_pointer= calculated pointer
       sthx  descendptr

GETDESC_END2:
       ldhx  descptr            ; get pointer to start of descriptor
       aix   #8                 ; eight bytes sent already
       sthx  descptr            ; store current pointer
       mov   #$88,UEP0CSR       ; SIZE=8 bytes, IN packet data ready
       mov   #GET_DESC,control  ; set flag for control transfer type
       bra   GETDESC_EXIT

GETDESC_STALL:
       jsr   FORCE_STALL

GETDESC_EXIT:
       rts

*****************************************************************
* SETUP_PROC - This procedure handles the SETUP packets that   *
*              come into the USB peripheral. The only standard *
```

```
*               device requests handled are SYNC_FRAME,         *
*               GET_DESCRIPTOR, and vendor-specific requests.   *
*****************************************************************
SETUP_PROC:
       clr   control            ; clear flag for control transfer
       lda   UEP0CSR            ; check size of SETUP packet
       lsra
       lsra
       lsra
       lsra
       cmp   #8                 ; is SIZE=8?
       bne   SETUP_EXIT         ; if not then exit
       ldhx  #8
SAVE_SETUP:
       lda   {UE0D0-1},x        ; save data to array
       sta   {Setup_packet-1},x ; Setup_packet holds info
       dbnzx SAVE_SETUP
       lda   bmReqType          ; if request type is standard
       and   #$60               ; then go handle standard request
       bne   SETUP_STALL        ; otherwise, force stall

STANDARD:
       lda   bRequest           ; get request type
       cbeqa #GET_DESC,GET_DESCR    ; if getting descriptors then
                                    ; ready the next IN packets for
                                    ; the descriptor information
       bra   SETUP_STALL        ; otherwise, force stall

GET_DESCR:
       jsr   GETDESC_PROC       ; packet received was GET_DESCRIPTOR
       bra   SETUP_EXIT         ; go exit

SETUP_STALL:
       jsr   FORCE_STALL        ; error, force stall on endpoint 0

SETUP_EXIT:
       lda  UEP0CSR             ; setup endpoint 0 for more packets
       and  #$fc
       sta  UEP0CSR
       bclr 5,USBSR             ; clear setup flags
       rts

*****************************************************************
* IN_PROC - This procedure handles the IN packets that come     *
*           into the USB peripheral through endpoint 0.         *
*****************************************************************
IN_PROC:
       lda   control            ; is this IN packet for data stage
```

```
       cmp   #GET_DESC          ;  of GET_DESCRIPTOR?
       bne   IN_PROC2
       ldhx  #0                 ; clear index into data buffer
       sthx  buffptr
IN_SEND_DATA:
       ldhx  descptr            ; get pointer to descriptor
       lda   0,x                ; get descriptor byte
       aix   #1
       sthx  descptr            ; save pointer to descriptor
       ldhx  buffptr            ; get index to data buffer
       sta   UE0D0,x            ; place byte in buffer
       aix   #1
       sthx  buffptr            ; save index to data buffer
       ldhx  descptr            ; is it last descriptor byte?
       cphx  descendptr
       beq   LAST_DESC
       ldhx  buffptr            ; is data buffer filled?
       cphx  #8
       beq   DATAFILLED         ;  go send data
       bra   IN_SEND_DATA       ; else continue

LAST_DESC:
       clr   control            ; if so then end of data transfer
DATAFILLED:
       lda   buffptr+1          ; set the size of the buffer
       asla                     ; shift to upper nibble
       asla
       asla
       asla
       ora   #$08               ; set DVALID_IN bit that data is ready
       sta   UEP0CSR
       rts

IN_PROC2:
       bclr  2,UEP0CSR          ; clear the TFRC_IN bit for further
packets
       rts

*****************************************************************
* OUT_PROC - This procedure handles the OUT packets that are   *
*            sent by the USB peripheral through endpoint 0.    *
*****************************************************************
OUT_PROC:
       lda  UEP0CSR              ; setup endpoint 0 for more packets
      and  #$fc                ; all OUT packets received for endpoint 0
      sta  UEP0CSR              ; should be for status stage of requests
       rts
```

```
************************************************************
* MAIN_INIT - This is the point where code starts executing  *
*             after a RESET.                                 *
************************************************************
MAIN_INIT:
       rsp
       clra
       clrx
       mov   #$01,CONFIG        ; URSTD=1 (USB reset=interrupt),
                                ; disable COP watchdog
       mov   #$FF,PULLCR        ; enable pullups for port B
       jsr   Init_USB           ; Initialize USB peripheral
       cli                      ; Allow interrupts to happen

main_loop:
       bra   main_loop

************************************************************
* USB_ENDP_ISR - USB Endpoint Interrupt Service Routine.     *
*           Interrupts here if packets are received or       *
*           transmitted on any endpoint.                     *
************************************************************
USB_ENDP_ISR:
       brclr 0,UEP0CSR,USB_ENDP_ISR2   ; Is it an OUT packet received?
       brclr 1,UEP0CSR,USB_ENDP_ISR2
       jsr   OUT_PROC                   ; handle OUT packet for EP0
       rti                              ; exit interrupt

USB_ENDP_ISR2:
      brclr 2,UEP0CSR,USB_ENDP_ISR3   ; Is packet sent from IN packet
for endpoint 0?
       jsr   IN_PROC                  ; handle IN packet for endpoint 0
       rti                             ; exit interrupt

USB_ENDP_ISR3:
      brclr 0,UEP1CSR,USB_ENDP_ISR4   ; Is packet sent from IN packet
for endpoint 1?
       lda   ptb                       ; get Port B value
       sta   EP1BuffStart             ; place in endpoint 1 data buffer
       mov   #01,UEP1DSR               ; set data size for endpoint 1
to 1
       bclr  0,UEP1CSR                 ; clear the TFRC flag
       bset  1,UEP1CSR                 ; set the DVALID flag

USB_ENDP_ISR4:
      brclr 0,UEP2CSR,USB_ENDP_EXIT   ; is it a packet for endpoint 2?
       brclr 1,UEP2CSR,USB_ENDP_EXIT   ; is it an OUT packet?
       ldhx  #0
```

```
USB_ISR4_LOOP:
       lda   EP2BuffStart,x           ; get data received in endpoint
buffer
       sta   EP2Data,x                ; and transfer to local data
buffer
       aix   #1
       cpx   UEP2DSR
       bne   USB_ISR4_LOOP
       lda   UEP2CSR                  ; clear the TFRC/DVALID flag
       and   #$fc
       sta   UEP2CSR
       rti                            ; exit interrupt

USB_ENDP_EXIT:
       rti

************************************************************
* USB_SYS_ISR - USB System Interrupt Service Routine.      *
*           Interrupts here if USB suspend, resume, reset, *
*           config_chg, start of frame, or setup event     *
*           occurs.                                        *
************************************************************
USB_SYS_ISR:
       brclr 2,USBSR,USB_SYS_ISR2   ; Is it USB reset?
       bclr  2,USBSR                ; reset USB reset flag
       rti                          ; exit interrupt

USB_SYS_ISR2:
       brclr 3,USBSR,USB_SYS_ISR3   ; Is there a change in
configuration?
       brclr 7,USBSR,USB_CONFIG_CLR
       lda   ptb                    ; get Port B value
       sta   EP1BuffStart           ; place in endpoint 1 data buffer
       mov   #01,UEP1DSR           ; set data size for endpoint 1 to 1
       bclr  0,UEP1CSR              ; clear the TFRC flag
       bset  1,UEP1CSR              ; set the DVALID flag
       bclr  3,USBSR                ; clear the CONFIG_CHG flag
       rti                          ; exit interrupt

USB_CONFIG_CLR:
       bclr  3,USBSR                ; clear CONFIG_CHG bit
       rti                          ; exit interrupt

USB_SYS_ISR3:
       brclr 5,USBSR,USB_SYS_ISR4   ; is it SETUP packet?
       jsr   SETUP_PROC             ; handle SETUP packet
       rti                          ; exit interrupt
```

```
USB_SYS_ISR4:
       brclr 0,USBSR,USB_SYS_EXIT   ; is it a SUSPEND event?
       bclr  0,USBSR                ; reset suspend flag
       rti                          ; exit interrupt

USB_SYS_EXIT:
       rti

*************************************************************
* DUMMY_ISR - Dummy Interrupt Service Routine.               *
*             Just does a return from interrupt.             *
*************************************************************
DUMMY_ISR:
       rti                      ; simple return

*************************************************************
* Vectors - Specifying Reset and USB Interrupt Routines    *
*************************************************************
       org   VectorStart

       dw    dummy_isr          ; TIM1 Overflow Vector
       dw    dummy_isr          ; TIM1 Channel 1 Vector
       dw    dummy_isr          ; TIM1 Channel 0 Vector
       dw    dummy_isr          ; PLL Vector
       dw    dummy_isr          ; IRQ Vector
       dw    USB_ENDP_ISR       ; USB Endpoint Vector
       dw    USB_SYS_ISR        ; USB System Vector
       dw    dummy_isr          ; SWI Vector
       dw    main_init          ; Reset Vector
```

This code utilizes three endpoints in the USB peripheral: endpoints 0, 1, and 2. As always, all control transactions occur through endpoint 0. The other endpoints can be set up for data transfer in the input or output direction, depending on whether data needs to be sent or received to/from the host. Here, endpoint 1 is setup as an input pipe and endpoint 2 is setup as an output pipe. The host becomes aware of whether an endpoint is an input or output through the descriptor tables that are exchanged at the beginning of device discovery.

It is necessary for the host and the slave to keep retention of the USB state of the device. There are multiple states for the USB device. The module starts in the POWERED state. When a USB reset is detected, the module is placed into the DEFAULT state. When the device receives a SET_ADDRESS standard device request with a valid address, the device is placed into the ADDRESSED state. Finally, when a SET_CONFIGURATION standard device request is sent, the device is placed into the CONFIGURED state.

After the device is configured, endpoint 1 is enabled for interrupt transfers from the host. Endpoint 1 is only capable of sending information out from it. Therefore, only IN packets

are accepted at the endpoint. The report descriptor sets up the endpoint to transfer only 1 byte of data. So when an IN packet comes from the host, the device sends out the data read from the Port B input pins. Note that the internal pullups are enabled for PORTB in the code (PULLCR), so the default input value sent through endpoint 1 should be $FF.

After the device is configured, endpoint 2 is enabled for bulk transfers. Endpoint 2 is only capable of reading information into it. Therefore, only OUT packets are accepted at the endpoint. The report descriptor sets up the endpoint to read, at most, 8 bytes of data. So when an OUT packet comes from the PC, the device reads the data from the packet into a local buffer.

So, how is the code laid out in order to handle this? The beginning of the code shows symbol defines for addresses within memory for RAM, flash, interrupt vectors, and buffers for USB endpoints. Also included are the register file definitions, which give the addresses of peripheral registers on the microcontroller. Following this comes the variable declarations for control packets, pointers, and received data.

In the FLASH, the descriptor tables are created, which describe the type of USB device, the configuration, the interface, HID specific information, the endpoints, and reporting information. All information needed in these descriptors, can be found in the Freescale documentation describing USB module specifications for the microprocessor that you are using.

After the descriptors comes the actual code, where numerous procedures are described, meant to handle different tasks. The first is the initialization procedure, which sets up the registers for the USB peripheral for the intended operation. The USB peripheral is also enabled at the end of the routine. The next several procedures are specifically for USB operation, such as: forcing a device stall, sending the device descriptors to the host, handling SETUP packets, handling IN packets, and processing OUT packets. After this comes the start of the application in the MAIN_INIT procedure, where we call the initialization routines and wait for an interrupt to occur. There are two interrupt service routines for the USB peripheral in this code: one for endpoint events that occur, and the other for USB system events such as setups, stalls, and resets. Note that this application is totally interrupt driven, where the code is driven by any events that take place in the USB peripheral.

The end of the code shows the interrupt vectors for the microcontroller, where the USB interrupts and the RESET vector are included. All other vectors are pointed to a dummy interrupt service routine that simply just returns from the interrupt.

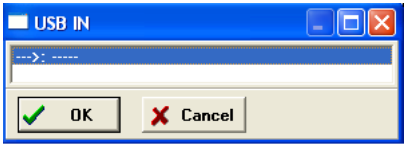
Once the mentioned above assembly code is compiled as a part of the Freescale project, one can step through in the Full Chip Simulation mode. The execution will automatically start at the reset location within the code. We are now ready to start debugging the code.

First, we need to step through the beginning of the code in the simulator in order to allow the code to initialize the microcontroller and the USB peripheral and get ready to start receiving USB packets from the host. So, the user needs to step through the beginning of the code until they get to the main loop with the “t” command.

At this point we are ready for USB packets to come into the device. There are four specific commands in the Full Chip Simulator for simulating external inpout/output as well as reset and clear functionality of the USB peripheral: USBRESET, USBIN, USBOUT, and USBCLR (for more detailed information, please refer to the USB commands section). The USBRESET command causes a USB reset, which is identical to a host trying to reset a slave USB device once it connects to the bus. The USBIN command allows a user to specify packets coming into the USB peripheral. The USBOUT command allows a user to see the packets that were sent out from the USB peripheral. The USBCLR command allows the user to clear all input and output buffers of USB packets. The first command that should be used for testing the USB peripheral is the USBRESET command, which places the USB peripheral in the RESET state. If the user steps after this with a “t” command, they will enter the system interrupt service routine. By entering the “t” command a few more times, the user will exit the ISR and return to the main loop.

Now we can use the USBIN command to specify USB packets that come across the USB bus from the host. When a user types this command in the simulator, the USB IN window appears as shown in Figure 11.56 on page 318. There are no USB packets specified yet in the window, and we will enter some into it.

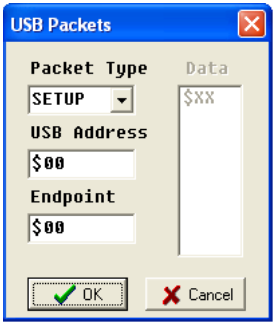
Figure 11.56 USB IN Buffer



Input packets can be added by double-clicking in the window where the packet is to go or by selecting the row for the packet and clicking the OK button. At this point a new window comes up as shown in Figure 11.57 on page 319. It is in this window that the user can specify the parameters of the packet. The “Packet Type” pull-down menu allows the user to select the type of USB packet, where it could be a SETUP, IN, OUT, DATA0, DATA1, ACK, NAK, STALL, or SOF packet. Note that if a SETUP, IN, or OUT packet is selected, then the DATA field is not utilized. If the DATA0 or DATA1 packet is selected, then the “USB Address” and “Endpoint” edit fields are not used. For SETUP, IN, and OUT packets, the “USB Address” and “Endpoint” fields must be filled with the appropriate data.

Here is an explanation of how to fill these fields for a GET DEVICE DESCRIPTOR request from the host, which typically comes right after the USB reset.

Figure 11.57 USB Packet Setup Dialog Box



The GET DEVICE DESCRIPTOR request between a host and a slave USB device has three different stages to it as defined by the USB standard: the setup stage, the data stage, and the status stage. Figure 11.58 on page 320 shows the flow of packets between the USB host and slave devices for a GET DEVICE DESCRIPTOR request as well as the stages. The setup stage simply consists of a SETUP packet from the host, then a DATA0 packet with the byte codes that indicate that this is a GET DESCRIPTOR, and the acknowledge (ACK) packet from the slave.

To replicate this stage in the simulator, the SETUP packet would be entered as shown in Figure 11.57 on page 319 and the DATA0 packet would be entered as shown in Figure 11.59 on page 321 with the USBIN command. The ACK is automatically sent out by the USB peripheral after the code services the other packets which are received in the simulator. Once the SETUP and DATA0 packets are entered, the USBIN window should look as shown in Figure 11.60 on page 321. The user can then step through the setup stage by typing “t” in the simulator. At this point the USBOUT command can be used to see that the ACK packet was sent out by the device, as shown in Figure 11.61 on page 321. This would complete the setup stage.

Just as we did in the setup stage, we can utilize the same commands for the data stage. So we can create IN and ACK packets with the USBIN command. Once the IN packet is received by the code, it sends out the descriptor information with data packets through endpoint 0 as it should. The ACK packet then is needed to end that data stage transfer. Several data stage transfers are needed in order to send out all of the device descriptor information. Each data packet sent out can hold only 8 bytes, so we need three transfers in this case. The USB IN window looks as shown in Figure 11.62 on page 321. The USB OUT window now looks different as well as shown in Figure 11.63 on page 322.

Figure 11.58 USB Packet Exchange Diagram

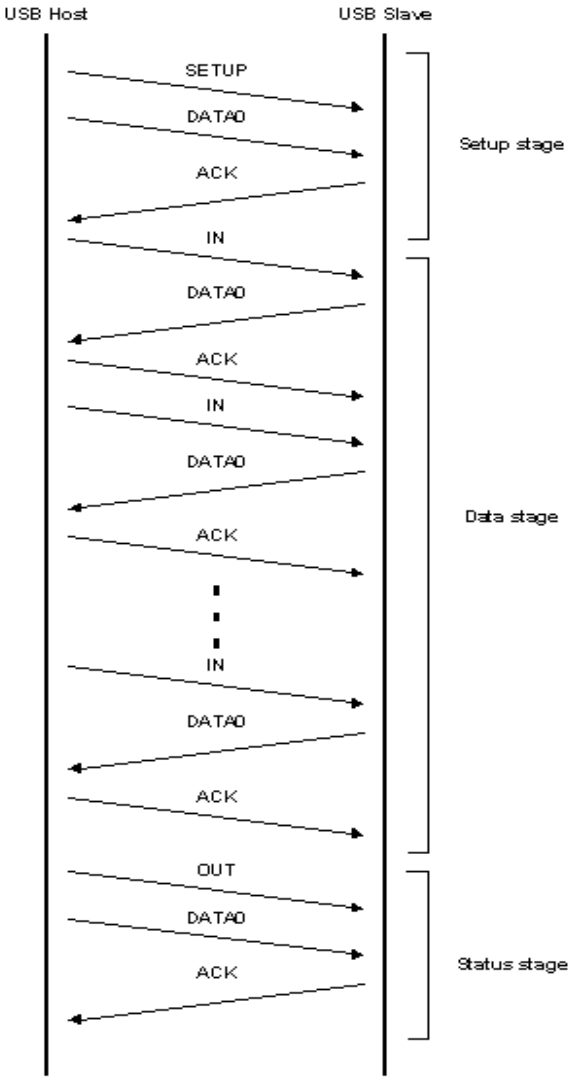


Figure 11.59 USB Packet Setup Full

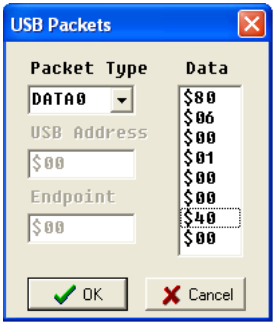


Figure 11.60 USB IN Data Buffer

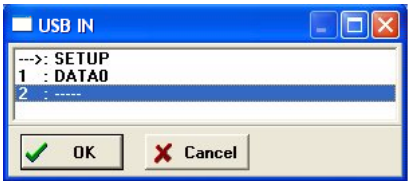


Figure 11.61 USB Out Buffer

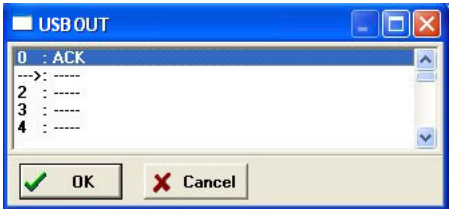
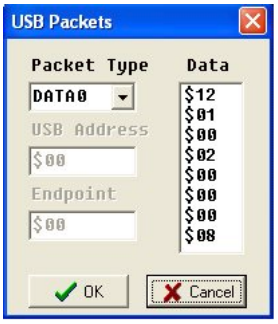


Figure 11.62 USB IN Buffer

Figure 11.63 USB OUT Buffer

Notice in Figure 11.64 on page 322 the DATA0 packet output during the data stage. This packet contains the data found in the device descriptor table in the code. This shows exactly what USB packet information was sent out from the device.

Figure 11.64 USB Packets IN Setup



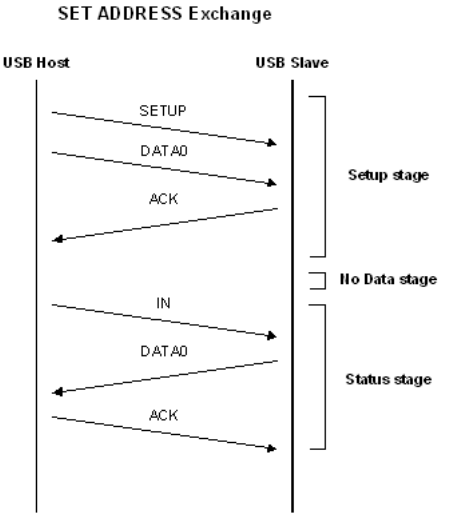
Once we get through the status stage, which requires an OUT packet along with an empty DATA0 packet from the host to be received, we get the USB IN window to look like that shown in Figure 11.65 on page 323 and the USB out window to look like that shown in Figure 11.66 on page 323

Figure 11.65 USB IN Buffer

Figure 11.66 USBOUT Buffer

Now that we have gone through this device descriptor exchange, the host has an idea of what type of USB slave device it is, and next needs to give it an address. We can go through the same process for the SET ADDRESS request from the host. Once the device is addressed, you must use this address in the SETUP packets sent to the device.

Figure 11.67 Set Address Packet Exchange



USB Commands

USBCLR Command

The USBCLR command can be used to flush the input and output buffers for USB simulation. This will reset the buffers and clear out all packets. Notice that if the USB is currently shifting a value, this command will not prevent the USB from finishing the transfer. See USBIN command and USBOUT command for accessing the input and output buffers of the USB interface.

Syntax:

```
>USBCLR
```

Example

```
>USBCLR
```

Clear input and output buffer for USB simulation

USBIN Command

The USBIN command allows the user to create packets for input into the USB. If packet parameters are given, the packet is placed into the next slot in the USB input buffer. Otherwise, if no parameter is given, a pick window is displayed with the input buffer

packets. Packets can be entered while the window is open. An arrow points to the packet that will be used next as input to the USB. The maximum number of input packets is 256.

Syntax:

```
USBIN [SETUP | IN | OUT <address> <endpoint>]
USBIN [DATA0 | DATA1 <n1>…<n2>…<n3>]
USBIN [ACK/NAK/STALL]
```

Where:

<n1>…<n2>…<n3>…The values for the data packet.

<address>The USB address for the packet

<endpoint>The endpoint number for the packet

Example:

```
>USB SETUP $5A $0
```

Set the next input packet as a SETUP packet for address $5A at the endpoint 0.

```
>USBIN DATA1 $10 $A4 $52
```

Set the next packet as a DATA1 packet with 3 data bytes $10, $A4, $52

```
>USBIN ACK
```

Set the next input packet as an ACK packet

```
>USBIN
```

Open the pick window with all the input packets

Where:

<n> The value to be entered into the next location in the input buffer

Example:

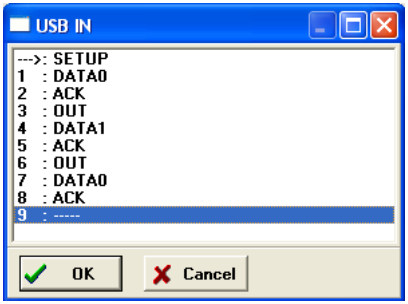
```
>USBIN $55
```

Set the next input value to the USB to $55

```
>USBIN
```

Pull up the data window with all the input values.

Figure 11.68 USB IN Buffer Display



USBOUT Command

The USBOUT command displays the output buffer from the USB. A window is opened that shows all the data that the USB has shifted out. An arrow is used to point to the last output packet transmitted. The maximum number of output packets that the buffer holds is 256 bytes.

Syntax:

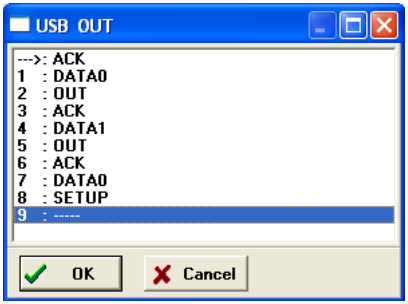
```
>USBOUT
```

Example:

```
>USBOUT
```

View packets from the output buffer for USB simulation.

Figure 11.69 USB OUT Buffer Display



USBRESET Command

The USB RESET command simulates a USB reset from the USB connection. If the URSTD bit is clear in the CONFIG register, then a reset of the MCU occurs just as with a RESET command. If the URSTD bit is set, then a USB interrupt occurs. See the technical description of the USB peripheral in the Freescale Manual for more information.

Syntax:

```
>USBRESET
```

Example:

```
>USBRESET
```

Simulate USB reset of the MCU.

12

MON08 Interface Connection

The MON08 connection setting permits a connection to Class 1-4 devices. Please see the descriptions below for a definition of each interface class.

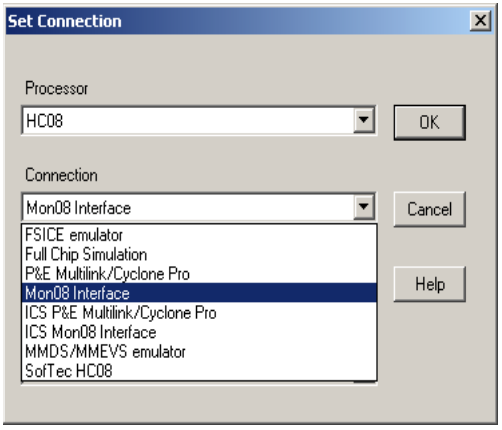
MON08 connection mode allows the user to debug code, as the firmware is fully resident in the FLASH of the microprocessor. The operation of all modules fully reflects the actual operation of the on-board resources.

Connection Procedure

To make the MON08 Interface debugger connection:

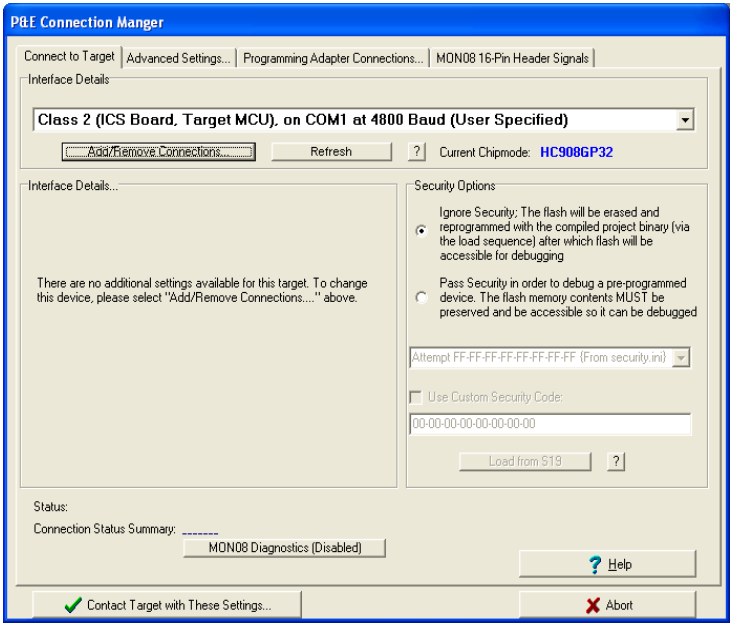
1. Choose the MON08 Interface option from the Set Connection dialog box

Figure 12.1 Set MON08 Interface Connection



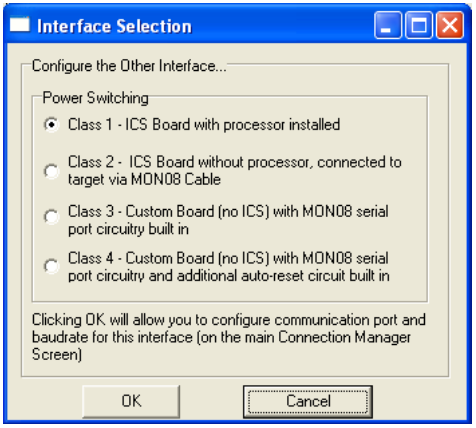
2. Click the OK button - The P&E Connection Manager window opens with its Connect Target tab selected.

Figure 12.2 P&E Connection Manager Window - Connect to Target Tab


P&E Connection Manger
Connect to Target
Advanced Settings...
Programming Adapter Connections...
MON08 16-Pin Header Signals
Interface Details
Class 2 (ICS Board, Target MCU), on COM1 at 4800 Baud (User Specified)
Add/Remove Connections...
Refresh
?
Current Chipmode:
HC908GP32
Interface Details...
There are no additional settings available for this target. To change this device, please select "Add/Remove Connections...." above.
Security Options
Ignore Security; The flash will be erased and reprogrammed with the compiled project binary (via the load sequence) after which flash will be accessible for debugging
Pass Security in order to debug a pre-programmed device. The flash memory contents MUST be preserved and be accessible so it can be debugged
Attempt FF-FF-FF-FF-FF-FF-FF-FF (From security.ini)
Use Custom Security Code:
00-00-00-00-00-00-00-00
Load from S19
?
Status:
Connection Status Summary:
MON08 Diagnostics (Disabled)
Help
Contact Target with These Settings...
Abort

3. Access the Interface Selection dialog box by clicking the “Add A Connection” button in the Connection Manager window’s - Connect Target Tab. Choose a device class corresponding to the type of interface that you are using.

Figure 12.3 Interface Selection Dialog Box

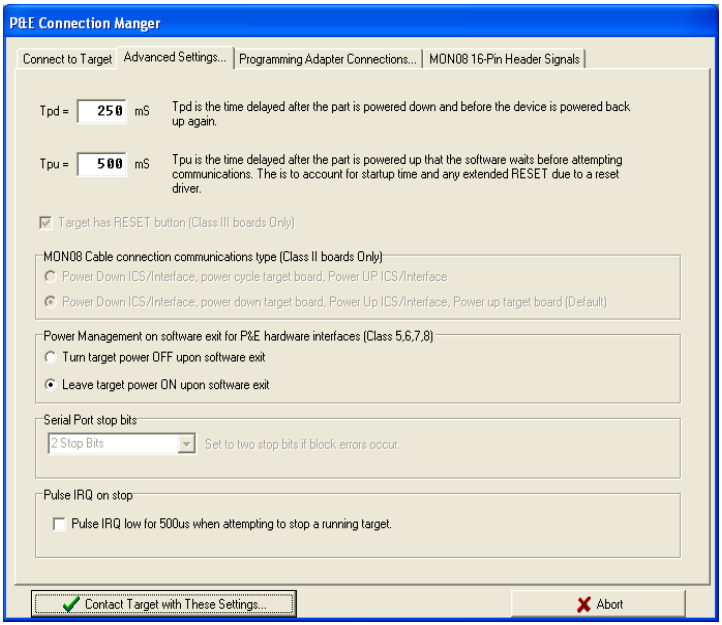

Interface Selection
Configure the Other Interface...
Power Switching
Class 1 - ICS Board with processor installed
Class 2 - ICS Board without processor, connected to target via MON08 Cable
Class 3 - Custom Board (no ICS) with MON08 serial port circuitry built in
Class 4 - Custom Board (no ICS) with MON08 serial port circuitry and additional auto-reset circuit built in
Clicking OK will allow you to configure communication port and baudrate for this interface (on the main Connection Manager Screen)
OK
Cancel

4. Define the proper communication port and baud rate setting in the Connection Manager.
5. To remove a pre-configured MON08 Interface connection, please proceed to the Remove section of the connection manager. Select the interface to be removed and click on “Remove Selected Interface.”

Advanced Settings Tab

The Advanced Settings tab allows the user to set specific protocol settings. The following is an explanation of each part of the Advanced Settings tab.

Figure 12.4 P&E Connection Manager Window - Advanced Settings Tab


P&E Connection Manger
Connect to Target
Advanced Settings...
Programming Adapter Connections...
MON08 16-Pin Header Signals
Tpd = 250 mS
Tpd is the time delayed after the part is powered down and before the device is powered back up again.
Tpu = 500 mS
Tpu is the time delayed after the part is powered up that the software waits before attempting communications. The is to account for startup time and any extended RESET due to a reset driver.
Target has RESET button (Class III boards Only)
MON08 Cable connection communications type (Class II boards Only)
Power Down ICS/Interface, power cycle target board, Power UP ICS/Interface
Power Down ICS/Interface, power down target board, Power Up ICS/Interface, Power up target board (Default)
Power Management on software exit for P&E hardware interfaces (Class 5,6,7,8)
Turn target power OFF upon software exit
Leave target power ON upon software exit
Serial Port stop bits
2 Stop Bits
Set to two stop bits if block errors occur.
Pulse IRQ on stop
Pulse IRQ low for 500us when attempting to stop a running target.
Contact Target with These Settings...
Abort

Tpd And Tpu Timing Listboxes

Tpu and Tpd Timing Listboxes set the power-up and power-down delay (respectively) that will be observed when power-cycling a target for entry into Monitor Mode. These settings are only valid for devices with automatically controlled power.

Whenever power is automatically switched off, or is manually requested to be switched off, the software waits for an amount of time equal to the Tpd delay time before proceeding to the connection protocol. This is because a board or power supply may have capacitance which holds the power up for a short time after the supply has been switched

off, but the supply voltage must reach less than 0.1v before it is turned back on if a Power-On reset is to occur.

Whenever power is automatically switched on, or is manually requested to be switched on, the software waits for an amount of time equal to the Tpu delay time before attempting to contact the 68HC08 processor. This is to allow time not only for power to be fully available, but to wait until any reset driver has finally released the RESET line.

Target Has RESET Button

With a class III board, the software occasionally needs to get control of the target. On systems which are Class III boards with the monitor mode circuitry built-in (including RS-232 driver), there is no means to reset the target to gain control. If the board has a reset button, the software can use this to gain control of the target system. If this option is checked, the software will prompt the user to push the target reset button when a reset of the target system is desired. If the option is unchecked, the software will ask the user to power cycle the target system to achieve a reset.

For a detailed description of the board or device classes, see on page 343“Device Class Description” on page 343.

MON08 Cable Connection Communications Type

This selection box is valid only for Class II hardware configurations using the MON08 cable. For a detailed description of the board or device classes, see on page 343“Device Class Description” on page 343. It allows the user to specify the sequence that the software uses to power up the ICS system. When the software tries to create a power-on reset condition, two events must occur:

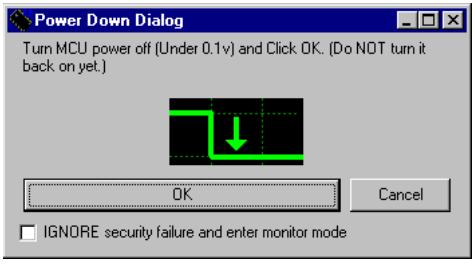
1. Power of the target MCU must go below 0.1v. This means that the processor can not be receiving power from its power pins, nor can it have a significant voltage being driven on port pins or the IRQ line, as these will drive the MCU power back through these pins. It is crucial, therefore, to have the ICS and the Target both powered down at some point in time.
2. The processor MON08 configuration pins, including IRQ, must be properly driven when the target processor resets to drive it into monitor mode. If these pins are not set up properly before the processor powers up, the processor may start up in user mode.

Power Up and Power Down Radio Buttons

This is the default option and should work for most, if not all, ICS08/Target Board solutions. Refer to the manual addendum under startup for the settings for a specific ICS board. It requires the user go through two dialog stages, and requires more time than simply cycling the power.

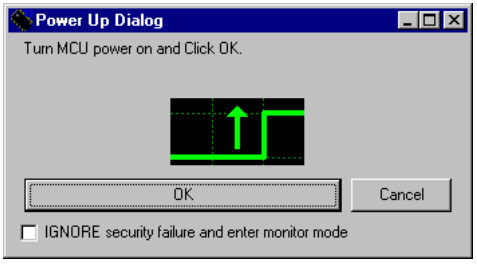
1. Software automatically powers down the ICS.
2. Software asks the user to power down the board as follows:

Figure 12.5 Power Down Dialog Box


Power Down Dialog
Turn MCU power off (Under 0.1v) and Click OK. (Do NOT turn it back on yet.)
OK
Cancel
IGNORE security failure and enter monitor mode

3. Software automatically powers up the ICS, which configures the processor's MON08 configuration pins.
4. Software asks the user to power up the board as follows:

Figure 12.6 Power Up Dialog Box

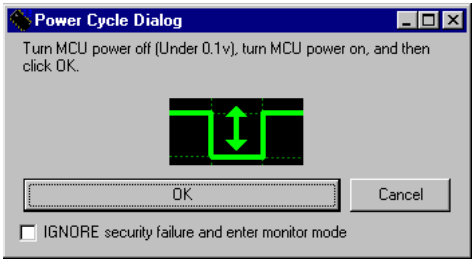


Turn Target Power Off and Leave Target Power On Radio Buttons

This option will work for many ICS boards as well, but relies on the fact that while the ICS is powered off, it will hold the target in reset until it is powered up itself and has configured the MON08 configuration pins. The sequence of events in this mode is:

1. Software automatically powers down the ICS.
2. Software asks the user to power cycle their board as follows:

Figure 12.7 Power Cycle Dialog Box


Power Cycle Dialog
Turn MCU power off (Under 0.1v), turn MCU power on, and then click OK.
OK
Cancel
IGNORE security failure and enter monitor mode

3. Software automatically powers up the ICS, which configures the processors MON08 configuration pins.

Serial Port Stop Bits

Serial Port Stop Bits allow users of Class 1-4 devices that are experiencing unreliable communication to increase the number of stop bits to 2.

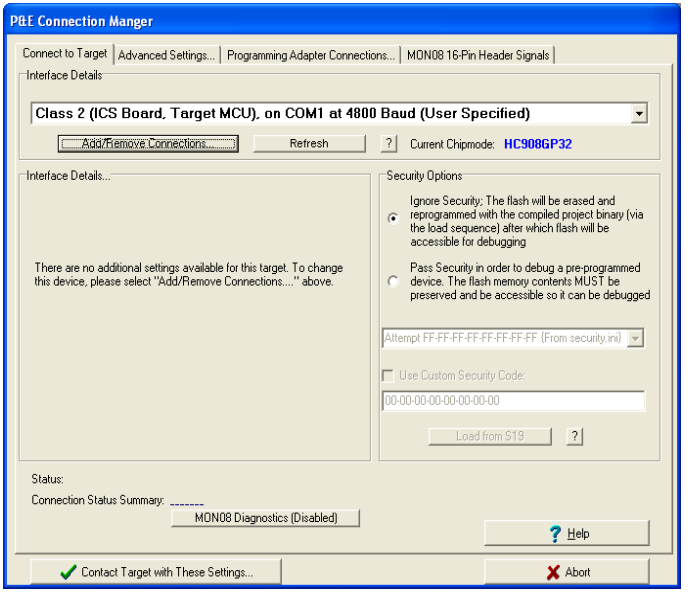
Pulse IRQ on Stop

The IRQ on Stop allows the users to execute a mechanism to stop a running HC08 microprocessor. For more information, please refer to the Stop a Running HC908 Target.pdf application note that is available from P&E Microcomputer Systems website: www.pemicro.com.

Target MCU Security Bytes

One of the steps that is necessary to properly bypass security is to provide the proper security code for the information that is programmed into the part. This holds true even when the part is blank. The security code consists of the 8 values which are currently stored in flash locations $FFF6 - $FFFD of the processor.

Figure 12.8 P&E Connection Manager Window - MON08 16-Pin Header Signals Tab


P&E Connection Manger
Connect to Target
Advanced Settings...
Programming Adapter Connections...
MON08 16-Pin Header Signals
Interface Details
Class 2 (ICS Board, Target MCU), on COM1 at 4800 Baud (User Specified)
Add/Remove Connections...
Refresh
?
Current Chipmode: HC908GP32
Interface Details...
There are no additional settings available for this target. To change this device, please select "Add/Remove Connections...." above.
Security Options
Ignore Security: The flash will be erased and reprogrammed with the compiled project binary (via the load sequence) after which flash will be accessible for debugging
Pass Security in order to debug a pre-programmed device. The flash memory contents MUST be preserved and be accessible so it can be debugged
Attempt FF-FF-FF-FF-FF-FF-FF-FF (From security.ini)
Use Custom Security Code:
00-00-00-00-00-00-00-00
Load from S19
?
Status:
Connection Status Summary:
MON08 Diagnostics (Disabled)
Help
Contact Target with These Settings...
Abort

The MON08 16-Pin Header Signals tab of the P&E Connections Manager window can be used by the user to manually enter the proper security bytes via the USER setting, or to load the security bytes from the same .S19 file which was programmed. The bytes are loaded from an .S19 file by clicking the “Load from S19” button.

IGNORE Security Failure And Enter Monitor Mode

This MON08 16-Pin Header Signals tab checkbox can be used to cause the software to ignore a failure to properly pass the 68HC08 security check. If the checkbox is set, the software will attempt to establish monitor mode communications regardless of the security status. As long as the Baud and Port are correct, and the device has been properly powered, this will allow monitor mode entry. By ignoring the security check failure you may use monitor mode, but the ROM/Flash will not be accessible.

NOTE If a connection is not established for a reason other than security failure, the connection dialog will always appear.

STATUS Area

The status area of the MON08 16-Pin Header Signals tab consists of one status string following the “Status:” label, and seven items which list the state of the last attempt to connect to a target and pass security. The description for these items is as follows:

0 – ICS Hardware loopback detected:

Every ICS or board which supports MON08 has a serial loopback in hardware which, by connecting the transmit and receive lines, automatically echoes characters from the PC. A valid character transmitted from the PC should be echoed once by the loopback circuitry on the board and once by the monitor of the target processor itself. This status indicates whether or not the first echoed character from the hardware loopback was received when one of the security bytes was transmitted. If the status is ‘N’, which indicates that the character was not received, it is most likely due to one of the following reasons:

1. Wrong Com Port specified.
2. The baud rate specified was incorrect (probably too low).
3. The ICS/Target is not connected.
4. No Power to the ICS.

If this status bit responded with an ‘N’, you must correct this before analyzing the reset of the status bits.

1 – Device echoed some security bytes:

The monitor resident in a 68HC08 device automatically echoes every incoming character when it is in monitor mode. A valid character transmitted from the PC should be echoed once by the loopback circuitry on the board and once by the monitor of the target processor itself. This status indicates whether or not the second echoed character from the monitor response was received when one of the security bytes was transmitted. If the status is ‘N’, which indicates that the character was not received, or not received properly, it is most likely due to one of the following reasons:

1. The baud rate specified was incorrect.
2. The part did not start the monitor mode security check on reset. Signals to force monitor mode may be incorrect.
3. No Power to the ICS.

If this status bit responded with an ‘N’, you must correct this before analyzing the reset of the status bits.

2 – Device echoed all security bytes:

In order to pass security, the software must send 8 security bytes to the processor. The processor should echo each of these eight bytes twice. If all 8 bytes did not get the proper two-byte echo, this flag will be 'N'. Reasons for this include:

1. The part did not start the monitor mode security check on reset. Signals to force monitor mode may be incorrect.
2. The baud rate specified was incorrect.
3. The processor was not reset properly. Check the "Target Hardware Type" and if you are connecting to a class II board, check the "MON08 cable communication connections type" in the "advanced settings" dialog.

3 – Device signaled monitor mode with a break:

Once the processor has properly received the 8 bytes from the PC software to complete its security check, it should transmit a break character to the PC signaling entry into monitor mode. This break should be sent regardless of whether the security check was successfully passed. If a break was not received from the processor, this flag will be 'N'. Reasons for this include:

1. The baud rate specified was incorrect.
2. The processor was not reset properly. Check the "Target Hardware Type". If you are connecting to a class II board, check the "MON08 cable communication connections type" in the "advanced settings" dialog.

4 – Device entered monitor mode:

Once the software has received, or failed to receive, a break from the processor, it attempts to communicate with the monitor running on the 68HC08 processor. It tries to read the monitor version number by issuing a monitor mode read. If the processor fails to respond properly to this command, this flag will be 'N'.

5 – Reset was Power-On Reset:

If the device properly entered monitor mode (4), the software will read the reset status register (RSR). This read does not affect the security sequence, and occurs purely for diagnostic reasons. The reset status register indicates the conditions under which the processor underwent the last reset. For the software to pass the security check properly, it MUST first cause the processor to undergo a Power-On Reset. The software reads the reset status register to determine if the last reset was indeed caused by power-on. The result of the reset status register is indicated in parentheses after the flag value. If the highest bit is not set then the reset was not a power on reset, and the flag will indicate 'N'. Reasons for this include:

- The processor did not power all the way down because power was being supplied to the processor through either the port pins, IRQ line, RESET line, or power pins.
- The voltage driven on the power pin of the processor did not go below 0.1 volts.

– The processor was not reset properly. Check the “Target Hardware Type”. If you are connecting to a class II board, check the “MON08 cable communication connections type” in the “advanced settings” dialog.

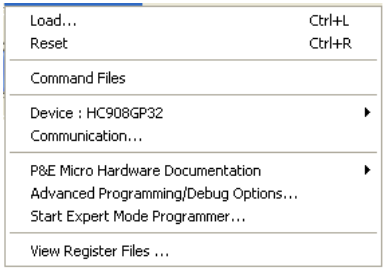
6 – ROM is accessible (un-secured):

If the device properly entered monitor mode (4), the software reads locations $FFF6-$FFFF to determine if the processor passes the security check. Memory locations which are invalid or protected read back from the device as $AD. If all bytes from $FFF6-$FFFF read a value of $AD, it is assumed the device is secure, and the flag value is an ‘N’. If all flags 0-5 register a value of ‘Y’ and flag 6 register a value of ‘N,’ then the reset process has gone correctly except that the security code used to pass security was incorrect. Specify the correct security code and try again, or IGNORE the security failure and erase the device. Once you erase a secured device, you must exit the software and restart it in order to pass security.

Active Mode Connection Menu Options

When the microprocessor is connected, more menu entries become available to the user.

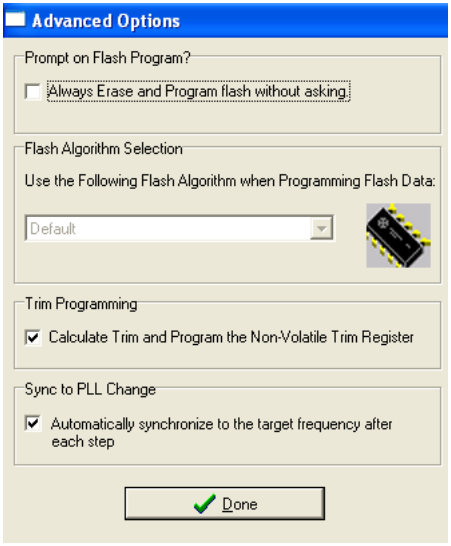
Figure 12.9 Additional MON08 Connection Menu Options



Advanced Programming/Debug Options Option

The Advanced Programming/Debug Options menu entry takes you to the Advanced Options dialog box, where you can configure the software settings for the FLASH programming procedure.

Figure 12.10 Advanced Options Dialog Box


Advanced Options
Prompt on Flash Program?
Always Erase and Program flash without asking.
Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
Default
Trim Programming
Calculate Trim and Program the Non-Volatile Trim Register
Sync to PLL Change
Automatically synchronize to the target frequency after each step
Done

Prompt on Flash Program Checkbox

Checking “Always Erase and Program flash without asking” lets the software transparently program the microprocessor.

Trim Programming Checkbox

The “Trim Programming” checkbox enables automatic calculation and programming of the trim value in a designated Non-Volatile memory location.

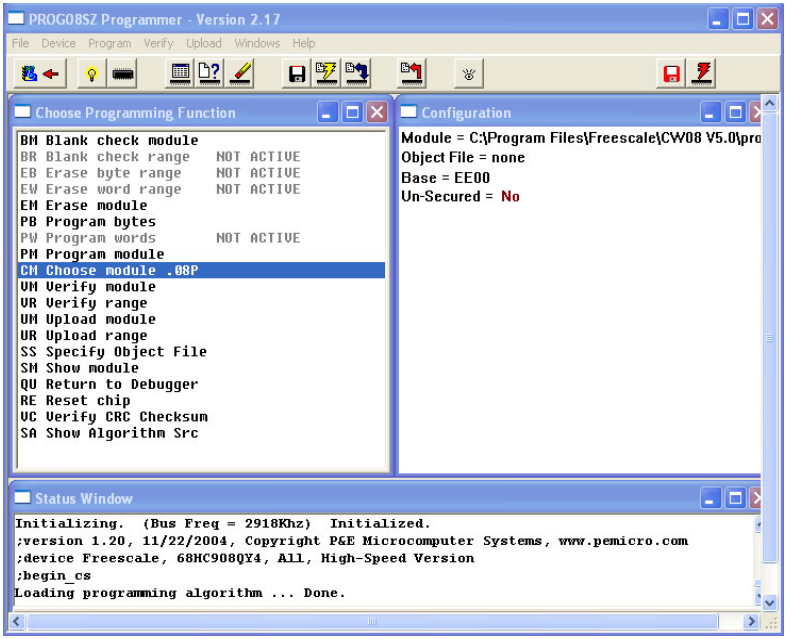
Sync to PLL Change Checkbox

“Sync to PLL Change” is required for the software/hardware connection to synchronize with the microprocessor during the flash erasing/programming procedure.

Start Expert Mode Programmer Option

“Start Expert Mode Programmer” grants the user access to P&E’s graphical flash programming utility, PROG08SZ. PROG08SZ lets an advanced user control the step-by-step execution of the flash erase/programming procedure. See Figure 12.11 on page 340. More information on how to use PROG08SZ can be found on P&E Microcomputer Systems website at: www.pemicro.com.

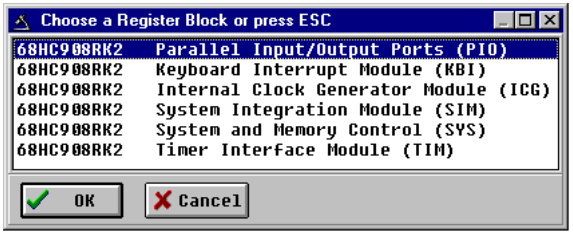
Figure 12.11 PROG08SZ Programmer Window


PROG08SZ Programmer - Version 2.17
File Device Program Verify Upload Windows Help
Choose Programming Function
BM Blank check module
BR Blank check range NOT ACTIVE
EB Erase byte range NOT ACTIVE
EW Erase word range NOT ACTIVE
EM Erase module
PB Program bytes
PW Program words NOT ACTIVE
PM Program module
CM Choose module .08P
VM Verify module
VR Verify range
UM Upload module
UR Upload range
SS Specify Object File
SM Show module
QU Return to Debugger
RE Reset chip
VC Verify CRC Checksum
SA Show Algorithm Src
Configuration
Module = C:\Program Files\Freescale\CW08 V5.0\pro
Object File = none
Base = EE00
Un-Secured = No
Status Window
Initializing. (Bus Freq = 2918Khz) Initialized.
;version 1.20, 11/22/2004, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Freescale, 68HC908QY4, All, High-Speed Version
;begin_cs
Loading programming algorithm ... Done.

View Register Files Option

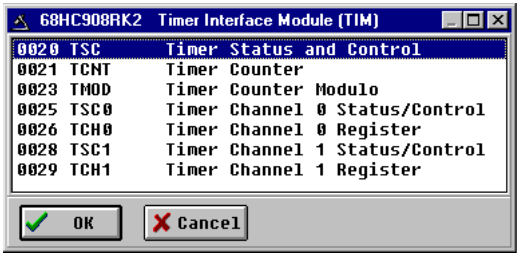
The “View Register Files...” menu option also gives the user the option of running the register file viewer/editor. If register files are available for the device that you have chosen, the “Choose a Register Block” window (see Figure 12.12 on page 341) will be opened. You may also open it by entering the R command in the Command Window command line.

Figure 12.12 Choose A Register Block Window


Choose a Register Block or press ESC
68HC908RK2 Parallel Input/Output Ports (PIO)
68HC908RK2 Keyboard Interrupt Module (KBI)
68HC908RK2 Internal Clock Generator Module (ICG)
68HC908RK2 System Integration Module (SIM)
68HC908RK2 System and Memory Control (SYS)
68HC908RK2 Timer Interface Module (TIM)
OK
Cancel

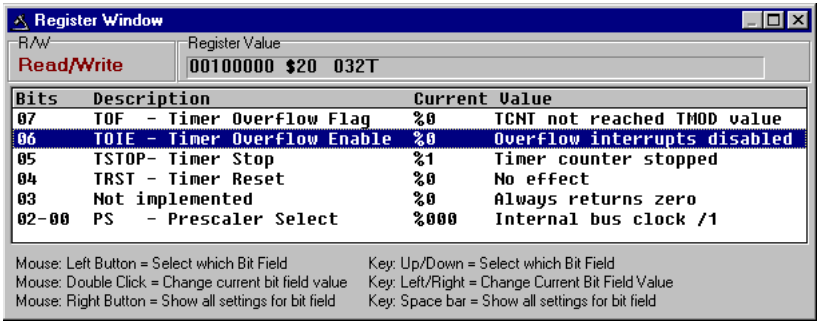
If register files have been installed on the host computer, selecting a block brings up the Register Block register listing (see Figure 12.13 on page 341), which shows a list of the associated registers, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.

Figure 12.13 Timer Interface Module Register Listing


68HC908RK2 Timer Interface Module (TIM)
0020 TSC Timer Status and Control
0021 TCNT Timer Counter
0023 TMOD Timer Counter Modulo
0025 TSC0 Timer Channel 0 Status/Control
0026 TCH0 Timer Channel 0 Register
0028 TSC1 Timer Channel 1 Status/Control
0029 TCH1 Timer Channel 1 Register
OK
Cancel

Selecting a file brings up the Register Window (see Figure 12.14 on page 342), which displays the values and significance for each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.

Figure 12.14 Register Window


Register Window
R/W
Read/Write
Register Value
00100000 $20 032T
Bits Description Current Value
07 TOF - Timer Overflow Flag %0 TCNT not reached TMOD value
06 TOIE - Timer Overflow Enable %0 Overflow interrupts disabled
05 TSTOP- Timer Stop %1 Timer counter stopped
04 TRST - Timer Reset %0 No effect
03 Not implemented %0 Always returns zero
02-00 PS - Prescaler Select %000 Internal bus clock /1
Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

Device Class Description

Class 1 Device

ICS Board with processor installed. This is the standard and most common configuration of the ICS08 boards. In this configuration, the processor is resident in one of the sockets on the ICS board itself. The processor can be debugged and programmed in this configuration, and an emulation cable containing all the processor I/O signals can be connected to the user's MCU development board. In this configuration, the ICS board hardware can automatically power up and down the processor in order to pass security in the simplest fashion. The user has to be sure not to provide power from the MCU development board, up through the emulation cable, to the processor pins themselves, when this dialog appears. This is so that the software, when attempting to establish communications, can fully power the processor down. The software running on the PC controls power to the user’s MCU development board via the serial port DTR line.

Class 2 Device

ICS Board without processor, connected to the user’s microprocessor system via MON08 Cable. In this configuration, there is no processor resident in any of the sockets of the ICS board itself. The processor is mounted down in the user’s MCU system. The connection from the ICS board to the user’s MCU system is accomplished via the 16-pin MON08 connector. In this configuration, since the ICS does not control power to the processor, the user will be prompted to turn the processor's power supply on and off. Turning off the power supply is necessary in order to be able to pass the initial security mode check and access the flash on the processor. A simple reset is not enough; to pass the security check, you must first force the processor to encounter a POR (power-on reset) which requires that the processor's voltage dip below 0.1v. Once security has been passed, resetting the device or re-entering the software should be easier.

Class 3 Device

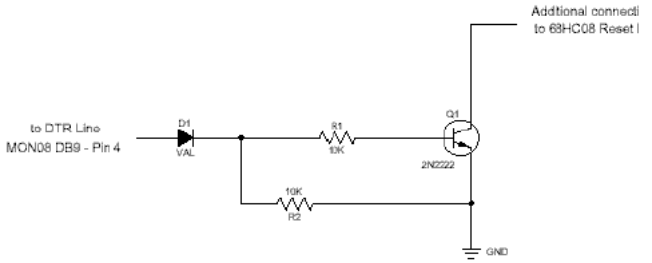
Custom Board (no ICS) with MON08 serial port circuitry built in. In this configuration, the ICS board is not used at all. The user must provide a serial port connection from the PC, and provide all hardware configuration necessary to force the processor into MON08 mode upon reset. This includes resets both internal and external to the processor. In this configuration, because the software does not directly control power to the processor, the user will be prompted to turn the processor's power supply on and off. The user will also be prompted to turn power on and off to reset the microprocessor, as the PC doesn't have control of the microprocessor reset. Turning off the power supply is necessary to be able to pass the initial security mode check and access the flash on the processor. A simple reset is not enough; to pass the security check, you must first force the processor to encounter a POR (power-on reset) which requires that the processor's voltage dip below

0.1v. Once security has been passed, resetting the device or re-entering the software should be easier. Class 3 selection also applies to use of the ICS board with the two-pin blank part programming connector.

Class 4 Device

Custom Board (no ICS) with MON08 serial port circuitry and additional auto-reset circuit built in. In this configuration, the ICS board is not used at all. The user must provide a serial port connection from the PC and all hardware configuration necessary to force the processor into MON08 mode upon reset. In addition, the user must include an extra circuit which allows the reset line of the processor to be driven low from the DTR line of the serial port connector (Pin 4 on a DB9). The following diagram shows the additional connection needed to reset from a DB9 serial connector.

Figure 12.15 Reset Connection



In this configuration, because the software does not directly control power to the processor, the user will be prompted to turn the processor's power supply on and off. Turning off the power supply is necessary in order to be able to pass the initial security mode check and access the flash on the processor. A simple reset is not enough; to pass the security check, you must first force the processor to encounter a POR (power-on reset) which requires the processor's voltage to dip below 0.1v. Once security has been passed, resetting the device should be facilitated by the above circuitry.

13

ICS MON08 Interface Connection

ICS Mode

In-Circuit Simulation (ICS) Mode is a P&E Microcomputer Systems mode of operation that is a hybrid between In-Circuit Debugging, and Full Chip simulation. P&E has combined the benefits of each of these modes, while minimizing their respective deficiencies.

In-Circuit Simulation, or ICS, mode simulates the CPU core instructions on the user's PC. However, Inputs/Outputs are read directly from the user's development device, and certain modules are run on the actual device. Any instructions that affect an I/O location-- for example, "lda PORTA" or "sta PORTA" -- would also use data from the real device. This allows for all the benefits of full chip simulation (Cycle by Cycle accuracy, unlimited breakpointing, and speed, to name just a few), while allowing the obvious benefit of garnering relevant signals from actual hardware.

In-Circuit simulation requires a user to have a connected MCU Device-- be it a development board, prototype hardware, or simply a device placed in a simple circuit on a breadboard. Once connection is established, the P&E Interface will load a .MON file to the microcontroller itself. This .MON file is the ICS-Kernel-- it constantly runs on the device, and interacts with the P&E software to provide information about changing I/O and any peripherals that are running on the device itself.

The following modules are run on the actual device when in ICS mode, and as such their Simulation commands become unnecessary, and therefore will not function:

I/O ports, SCI, SPI, IRQ, ADC, LCD, KBI.

All other modules will function as fully simulated modules. For information about relevant simulation commands, please see the "HC08 Full Chip Simulation Mode" section above.

Connection Procedure

To select the ICS MON08 Interface connection:

1. Choose the ICS MON08 Interface option from the set connection dialog box, as shown in Figure 13.1 on page 346.
2. Click the OK button to open the P&E Connection Manager Window.

Figure 13.1 ICS MON08 Set Connection Dialog Box

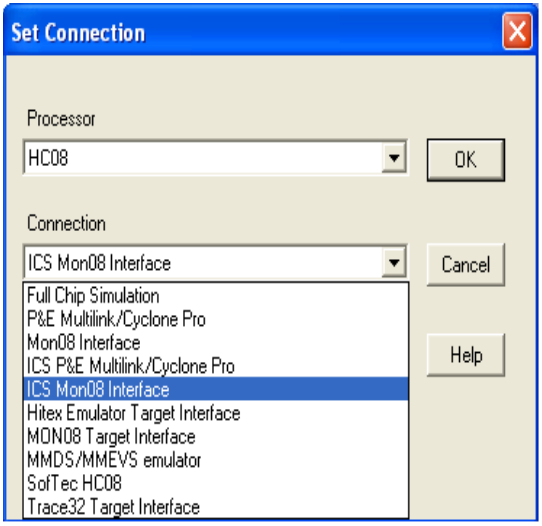
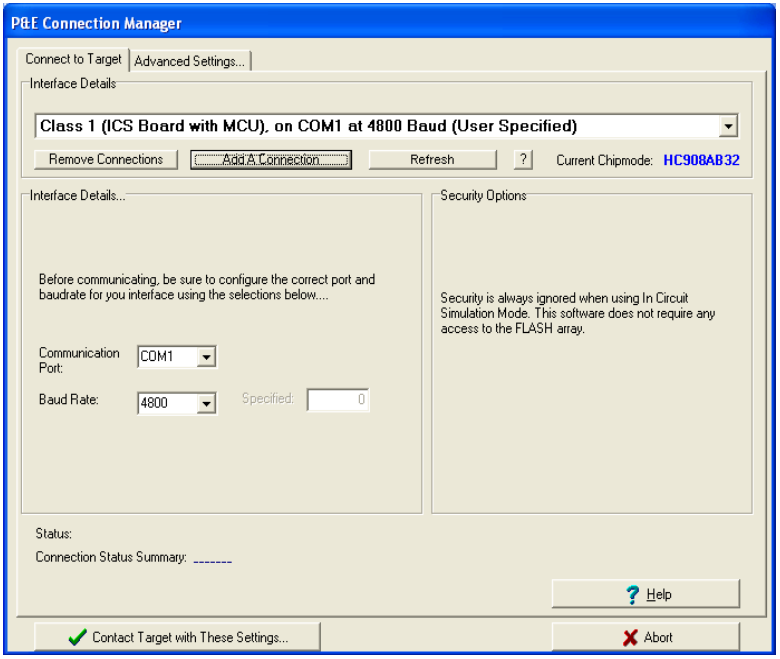
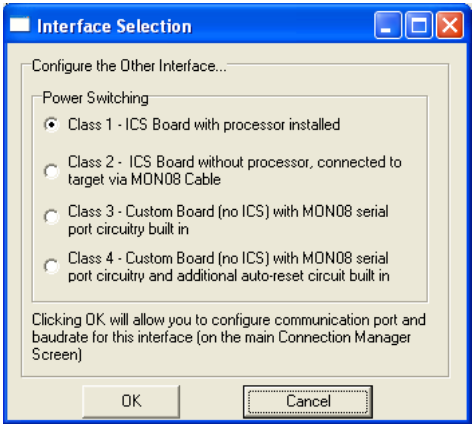


Figure 13.2 P&E Connection Manager Window - Connect to Target Tab



3. Access the Interface Selection Manager assistant by clicking on the “Add A Connection” button in the P&E Connection Manager Window (Figure 13.2 on page 347). Choose a device class that corresponds to the Interface that you are using.

Figure 13.3 Interface Selection Dialog Box

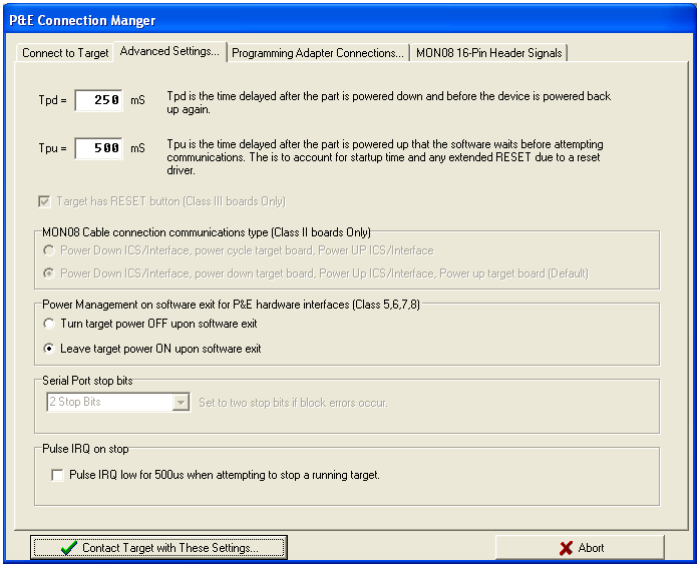

Interface Selection
Configure the Other Interface...
Power Switching
Class 1 - ICS Board with processor installed
Class 2 - ICS Board without processor, connected to target via MON08 Cable
Class 3 - Custom Board (no ICS) with MON08 serial port circuitry built in
Class 4 - Custom Board (no ICS) with MON08 serial port circuitry and additional auto-reset circuit built in
Clicking OK will allow you to configure communication port and baudrate for this interface (on the main Connection Manager Screen)
OK
Cancel

4. Define the proper communication port and baud rate setting in the Connection Manager. See Figure 13.3 on page 347.
5. To remove a pre-configured ICS MON08 Interface, go to the Remove section of the connection manager. Select the interface to be removed and click on “Remove Selected Interface.”

Advanced Settings Tab

The P&E Connection Manager window’s Advanced Settings tab allows the user to set specific protocol settings. The following is an explanation of each part of the advanced settings tab.

Figure 13.4 P&E Connection Manager Window - Advanced Settings Tab


P&E Connection Manger
Connect to Target
Advanced Settings...
Programming Adapter Connections...
MON08 16-Pin Header Signals
Tpd = 250 mS
Tpd is the time delayed after the part is powered down and before the device is powered back up again.
Tpu = 500 mS
Tpu is the time delayed after the part is powered up that the software waits before attempting communications. The is to account for startup time and any extended RESET due to a reset driver.
Target has RESET button (Class III boards Only)
MON08 Cable connection communications type (Class II boards Only)
Power Down ICS/Interface, power cycle target board, Power UP ICS/Interface
Power Down ICS/Interface, power down target board, Power Up ICS/Interface, Power up target board (Default)
Power Management on software exit for P&E hardware interfaces (Class 5,6,7,8)
Turn target power OFF upon software exit
Leave target power ON upon software exit
Serial Port stop bits
2 Stop Bits
Set to two stop bits if block errors occur.
Pulse IRQ on stop
Pulse IRQ low for 500us when attempting to stop a running target.
Contact Target with These Settings...
Abort

Tpd And Tpu Timing Textboxes

Tpu and Tpd will set the power-up and power-down delay (respectively) that will be observed when power-cycling a target for entry into Monitor Mode. These settings are only valid for devices with automatically controlled power.

Whenever power is automatically switched off, or is manually requested to be switched off, the software waits for an amount of time equal to the Tpd delay time before proceeding to the connection protocol. This is because a board or power supply may have capacitance which holds the power up for a short time after the supply has been switched

off, but the supply voltage must reach less than 0.1v before it is turned back on if a Power-On reset is to occur.

Whenever power is automatically switched on, or is manually requested to be switched on, the software waits for an amount of time equal to the Tpu delay time before attempting to contact the 68HC08 processor. This is to allow time not only for power to be fully available, but to wait until any reset driver has finally released the RESET line.

Target Has RESET Button

In Class III boards, the software occasionally needs to get control of the target. For a detailed definition of board classes, see on page 358Device Class Description on page 358. On systems which are Class III boards with the monitor mode circuitry built-in (including RS-232 driver), there is no means to reset the target to gain control. If the board has a reset button, the software can use this to gain control of the target system. If this option is checked, the software will prompt the user to push the target reset button when a reset of the target system is desired. If the option is unchecked, the software will ask the user to power cycle the target system to achieve a reset.

MON08 Cable Connection Communications Type

This selection box is valid only for Class II hardware configurations using the MON08 cable. For a detailed definition of board classes, see on page 358Device Class Description on page 358. It allows the user to specify the sequence that the software uses to power up the ICS system. When the software tries to create a power-on reset condition, two events must occur:

1. Power of the target MCU must go below 0.1v. This means that the processor can not be receiving power from its power pins, nor can it have a significant voltage being driven on port pins or the IRQ line, as these will drive the MCU power back through these pins. It is crucial, therefore, to have the ICS and the Target both powered down at some point in time.
2. The processor MON08 configuration pins, including IRQ, must be properly driven when the target processor resets to drive it into monitor mode. If these pins are not set up properly before the processor powers up, the processor may start up in user mode.

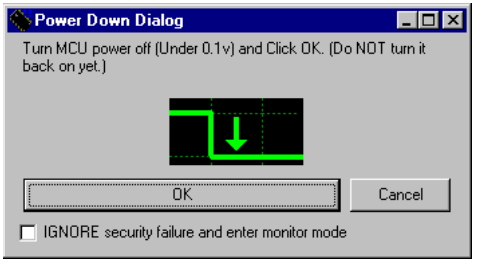
Power Up and Power Down Radio Buttons

Power Down ICS button asks the user to power down the board, Power Up ICS button asks the user to power up their board

This is the default option and should work for most, if not all, ICS08/Target Board solutions. Refer to the ICS board manual addendum under startup for the settings for a specific ICS board. It requires the user go through two dialog stages, and requires more time than simply cycling the power.

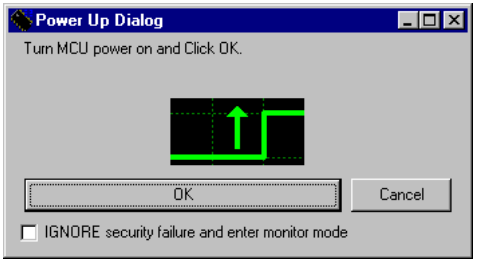
1. Software automatically powers down the ICS.
2. Software asks the user to power down the board as follows:

Figure 13.5 Power Down Dialog Box


Power Down Dialog
Turn MCU power off (Under 0.1v) and Click OK. (Do NOT turn it back on yet.)
OK
Cancel
IGNORE security failure and enter monitor mode

3. Software automatically powers up the ICS, which configures the processor's MON08 configuration pins.
4. Software asks the user to power up the board as follows:

Figure 13.6 Power Up Dialog Box



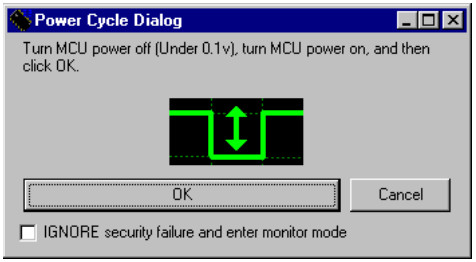
Cycle Power Radio Button

Power Down ICS button asks the user to power cycle their board. Power UP ICS button does the opposite.

This option will work for many ICS boards as well, but relies on the fact that while the ICS is powered off, it will hold the target in reset until it is powered up itself and has configured the MON08 configuration pins. The sequence of events in this mode is:

1. Software automatically powers down the ICS.
2. Software asks the user to power cycle their board as follows:

Figure 13.7 Power Cycle Dialog Box


Power Cycle Dialog
Turn MCU power off (Under 0.1v), turn MCU power on, and then click OK.
OK
Cancel
IGNORE security failure and enter monitor mode

3. Software automatically powers up the ICS, which configures the processors MON08 configuration pins.

Serial Port Stop Bits

Serial Port Stop Bits allow users of Class 1-4 devices that are experiencing unreliable communication to increase the number of stop bits to 2.

Pulse IRQ on Stop

The IRQ on Stop allows the users to execute a mechanism to stop a running HC08 microprocessor. For more information, please refer to the Stop a Running HC908 Target.pdf application note that is available from P&E Microcomputer Systems website: www.pemicro.com.

Target MCU Security Bytes

In ICS mode, the user's code is simulated on the PC, and not run from the Non-Volatile memory of the device. Therefore, there is no need to pass security in this mode of operation.

IGNORE Security Failure And Enter Monitor Mode

This checkbox can be used to cause the software to ignore a failure to properly pass the 68HC08 security check. If the checkbox is set, the software will attempt to establish monitor mode communications regardless of the security status. As long as the Baud and Port are correct, and the device has been properly powered, this will allow monitor mode entry. By ignoring the security check failure you may use monitor mode, but the ROM/ Flash will not be accessible.

NOTE If a connection is not established for a reason other than security failure, the connection dialog will always appear.

STATUS Area

The status area of the P&E Connections Manager window consists of one status string following the “Status:” label, and seven items which list the state of the last attempt to connect to a target and pass security. The description for these items is as follows:

0 – ICS Hardware loopback detected:

Every ICS or board which supports MON08 has a serial loopback in hardware which, by connecting the transmit and receive lines, automatically echoes characters from the PC. A valid character transmitted from the PC should be echoed once by the loopback circuitry on the board and once by the monitor of the target processor itself. This status indicates whether or not the first echoed character from the hardware loopback was received when one of the security bytes was transmitted. If the status is ‘N’, which indicates that the character was not received, it is most likely due to one of the following reasons:

– Wrong Com Port specified.
– The baud rate specified was incorrect (probably too low).
– The ICS/Target is not connected.
– No Power to the ICS.

If this status bit responded with an ‘N’, you must correct this before analyzing the reset of the status bits.

1 – Device echoed some security bytes:

The monitor resident in a 68HC08 device automatically echoes every incoming character when it is in monitor mode. A valid character transmitted from the PC should be echoed once by the loopback circuitry on the board and once by the monitor of the target processor itself. This status indicates whether or not the second echoed character from the monitor response was received when one of the security bytes was transmitted. If the status is ‘N’, which indicates that the character was not received, or not received properly, it is most likely due to one of the following reasons:

– The baud rate specified was incorrect.
– The part did not start the monitor mode security check on reset. Signals to force monitor mode may be incorrect.
– No Power to the ICS.

If this status bit responded with an ‘N’, you must correct this before analyzing the reset of the status bits.

2 – Device echoed all security bytes:

In order to pass security, the software must send 8 security bytes to the processor. The processor should echo each of these eight bytes twice. If all 8 bytes did not get the proper two-byte echo, this flag will be ‘N’. Reasons for this include:

– The part did not start the monitor mode security check on reset. Signals to force monitor mode may be incorrect.

– The baud rate specified was incorrect.

– The processor was not reset properly. Check the “Target Hardware Type” and if you are connecting to a class II board, check the “MON08 cable communication connections type” in the “advanced settings” dialog.

3 – Device signaled monitor mode with a break:

Once the processor has properly received the 8 bytes from the PC software to complete its security check, it should transmit a break character to the PC signaling entry into monitor mode. This break should be sent regardless of whether the security check was successfully passed. If a break was not received from the processor, this flag will be ‘N’. Reasons for this include:

1. The baud rate specified was incorrect.
2. The processor was not reset properly. Check the “Target Hardware Type”. If you are connecting to a class II board, check the “MON08 cable communication connections type” in the “advanced settings” dialog.

4 – Device entered monitor mode:

Once the software has received, or failed to receive, a break from the processor, it attempts to communicate with the monitor running on the 68HC08 processor. It tries to read the monitor version number by issuing a monitor mode read. If the processor fails to respond properly to this command, this flag will be ‘N’.

5 – Reset was Power-On Reset:

If the device properly entered monitor mode (4), the software will read the reset status register (RSR). This read does not affect the security sequence, and occurs purely for diagnostic reasons. The reset status register indicates the conditions under which the processor underwent the last reset. For the software to pass the security check properly, it MUST first cause the processor to undergo a Power-On Reset. The software reads the reset status register to determine if the last reset was indeed caused by power-on. The result of the reset status register is indicated in parentheses after the flag value. If the highest bit is not set then the reset was not a power on reset, and the flag will indicate ‘N’. Reasons for this include:

– The processor did not power all the way down because power was being supplied to the processor through either the port pins, IRQ line, RESET line, or power pins.

– The voltage driven on the power pin of the processor did not go below 0.1 volts.

– The processor was not reset properly. Check the “Target Hardware Type”. If you are connecting to a class II board, check the “MON08 cable communication connections type” in the “advanced settings” dialog.

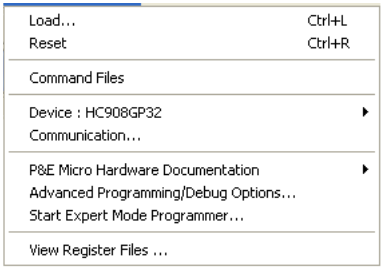
6 – ROM is accessible (un-secured):

If the device properly entered monitor mode (4), the software reads locations $FFF6-$FFFF to determine if the processor passes the security check. Memory locations which are invalid or protected read back from the device as $AD. If all bytes from $FFF6-$FFFF read a value of $AD, it is assumed the device is secure, and the flag value is an ‘N’. If all flags 0-5 register a value of ‘Y’ and flag 6 register a value of ‘N,’ then the reset process has gone correctly except that the security code used to pass security was incorrect. Specify the correct security code and try again, or IGNORE the security failure and erase the device. Once you erase a secured device, you must exit the software and restart it in order to pass security.

Active Mode Connection Menu Options

When the microprocessor is connected, more Connection menu entries become available to the user.

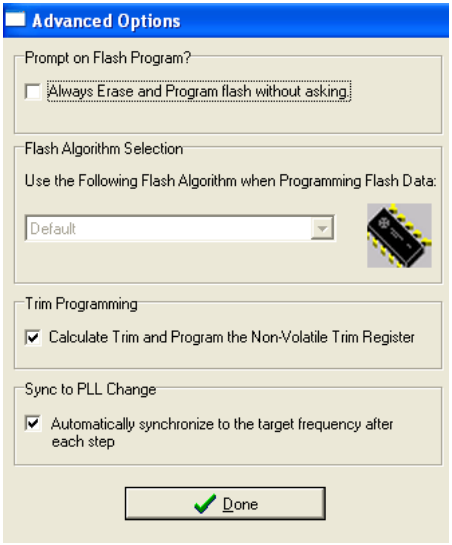
Figure 13.8 Additional Connection Menu Options



Advanced Programming/Debug Options Option

The Advanced Programming/Debug Options menu entry take you to the Advanced Options dialog box, where you can configure the software settings for the FLASH programming procedure.

Figure 13.9 Advanced Options Dialog Box


Advanced Options
Prompt on Flash Program?
Always Erase and Program flash without asking
Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
Default
Trim Programming
Calculate Trim and Program the Non-Volatile Trim Register
Sync to PLL Change
Automatically synchronize to the target frequency after each step
Done

Prompt on Flash Program Checkbox

Checking “Always Erase and Program flash without asking” lets the software transparently program the microprocessor.

Trim Programming Checkbox

The “Trim Programming” checkbox enables automatic calculation and programming of the trim value in a designated Non-Volatile memory location.

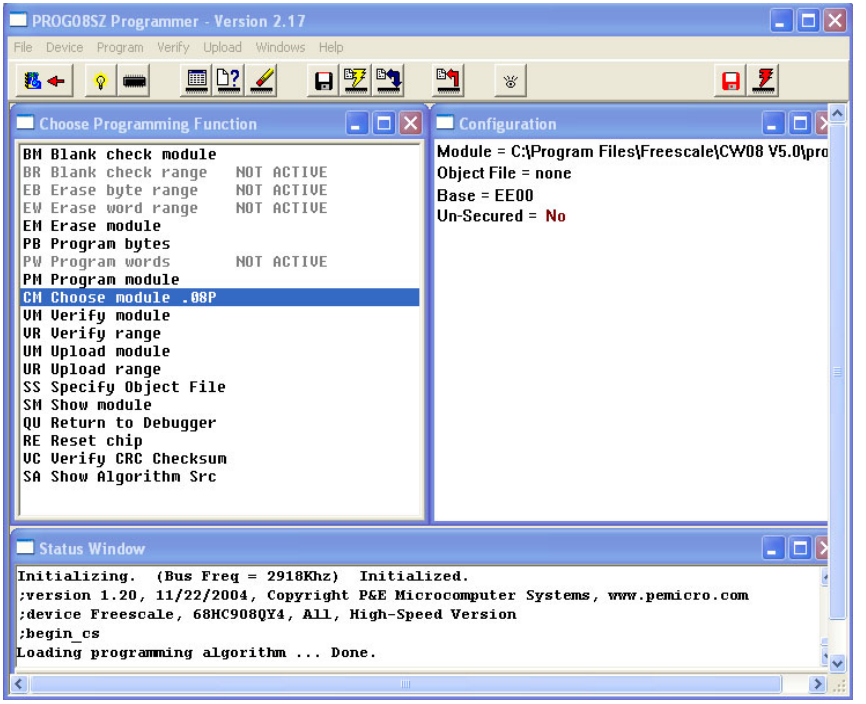
Sync to PLL Change Checkbox

“Sync to PLL Change” is required for the software/hardware connection to synchronize with the microprocessor during the flash erasing/programming procedure.

Start Expert Mode Programmer Option

“Start Expert Mode Programmer” grants the user access to P&E’s graphical flash programming utility, PROG08SZ. PROG08SZ lets an advanced user control the step-by-step execution of the flash erase/programming procedure. See Figure 13.10 on page 356. More information on how to use PROG08SZ can be found on P&E Microcomputer Systems website at: www.pemicro.com.

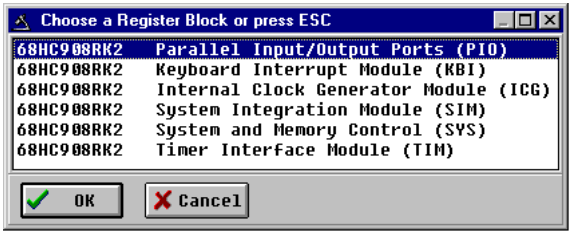
Figure 13.10 PROG08SZ Programmer Window


PROG08SZ Programmer - Version 2.17
File Device Program Verify Upload Windows Help
Choose Programming Function
BM Blank check module
BR Blank check range NOT ACTIVE
EB Erase byte range NOT ACTIVE
EW Erase word range NOT ACTIVE
EM Erase module
PB Program bytes
PW Program words NOT ACTIVE
PM Program module
CM Choose module .08P
VM Verify module
VR Verify range
UM Upload module
UR Upload range
SS Specify Object File
SM Show module
QU Return to Debugger
RE Reset chip
VC Verify CRC Checksum
SA Show Algorithm Src
Configuration
Module = C:\Program Files\Freescale\CW08 V5.0\pro
Object File = none
Base = EE00
Un-Secured = No
Status Window
Initializing. (Bus Freq = 2918Khz) Initialized.
;version 1.20, 11/22/2004, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Freescale, 68HC908QY4, All, High-Speed Version
;begin_cs
Loading programming algorithm ... Done.

View Register Files Option

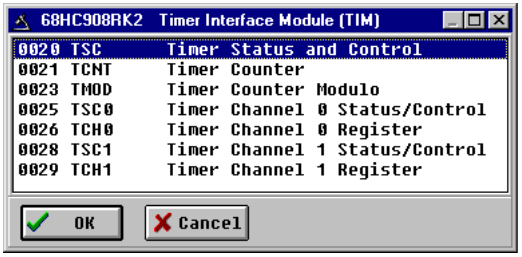
The “View Register Files...” menu option also gives the user the option of running the register file viewer/editor. If register files are available for the device that you have chosen, the “Choose a Register Block” window (see Figure 13.11 on page 357) will be opened. You may also open it by entering the R command in the Command Window command line.

Figure 13.11 Choose A Register Block Dialog Box


Choose a Register Block or press ESC
68HC908RK2 Parallel Input/Output Ports (PIO)
68HC908RK2 Keyboard Interrupt Module (KBI)
68HC908RK2 Internal Clock Generator Module (ICG)
68HC908RK2 System Integration Module (SIM)
68HC908RK2 System and Memory Control (SYS)
68HC908RK2 Timer Interface Module (TIM)
OK
Cancel

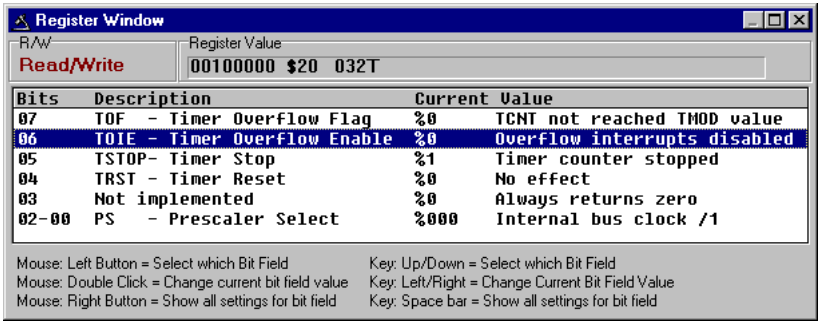
If register files have been installed on the host computer, selecting a block brings up the Register Block register listing (see Figure 13.12 on page 357), which shows a list of the associated registers, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.

Figure 13.12 Timer Interface Module Register Listing


68HC908RK2 Timer Interface Module (TIM)
0020 TSC Timer Status and Control
0021 TCNT Timer Counter
0023 TMOD Timer Counter Modulo
0025 TSC0 Timer Channel 0 Status/Control
0026 TCH0 Timer Channel 0 Register
0028 TSC1 Timer Channel 1 Status/Control
0029 TCH1 Timer Channel 1 Register
OK
Cancel

Selecting a file brings up the Register Window (see Figure 13.13 on page 357), which displays the values and significance for each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.

Figure 13.13 Register Window


Register Window
R/W
Read/Write
Register Value
00100000 $20 032T
Bits Description Current Value
07 TOF - Timer Overflow Flag %0 TCNT not reached TMOD value
06 TOIE - Timer Overflow Enable %0 Overflow interrupts disabled
05 TSTOP- Timer Stop %1 Timer counter stopped
04 TRST - Timer Reset %0 No effect
03 Not implemented %0 Always returns zero
02-00 PS - Prescaler Select %000 Internal bus clock /1
Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

Device Class Description

Class 1 Device

ICS Board with processor installed. This is the standard and most common configuration of the ICS08 boards. In this configuration, the processor is resident in one of the sockets on the ICS board itself. The processor can be debugged and programmed in this configuration, and an emulation cable containing all the processor I/O signals can be connected to the user's MCU development board. In this configuration, the ICS board hardware can automatically power up and down the processor in order to pass security in the simplest fashion. The user has to be sure not to provide power from the MCU development board, up through the emulation cable, to the processor pins themselves, when this dialog appears. This is so that the software, when attempting to establish communications, can fully power the processor down. The software running on the PC controls power to the user’s MCU development board via the serial port DTR line.

Class 2 Device

ICS Board without processor, connected to the user’s microprocessor system via MON08 Cable. In this configuration, there is no processor resident in any of the sockets of the ICS board itself. The processor is mounted down in the user’s MCU system. The connection from the ICS board to the user’s MCU system is accomplished via the 16-pin MON08 connector. In this configuration, since the ICS does not control power to the processor, the user will be prompted to turn the processor's power supply on and off. Turning off the power supply is necessary in order to be able to pass the initial security mode check and access the flash on the processor. A simple reset is not enough; to pass the security check, you must first force the processor to encounter a POR (power-on reset) which requires that the processor's voltage dip below 0.1v. Once security has been passed, resetting the device or re-entering the software should be easier.

Class 3 Device

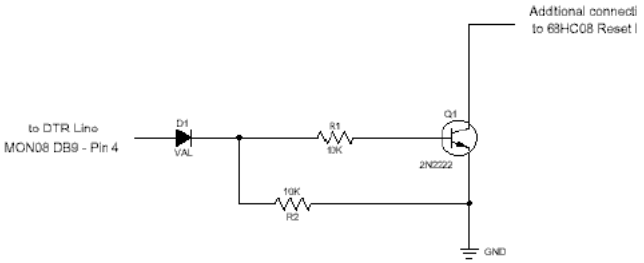
Custom Board (no ICS) with MON08 serial port circuitry built in. In this configuration, the ICS board is not used at all. The user must provide a serial port connection from the PC, and provide all hardware configuration necessary to force the processor into MON08 mode upon reset. This includes resets both internal and external to the processor. In this configuration, because the software does not directly control power to the processor, the user will be prompted to turn the processor's power supply on and off. The user will also be prompted to turn power on and off to reset the microprocessor, as the PC doesn't have control of the microprocessor reset. Turning off the power supply is necessary to be able to pass the initial security mode check and access the flash on the processor. A simple reset is not enough; to pass the security check, you must first force the processor to encounter a POR (power-on reset) which requires that the processor's voltage dip below

0.1v. Once security has been passed, resetting the device or re-entering the software should be easier. Class 3 selection also applies to use of the ICS board with the two-pin blank part programming connector.

Class 4 Device

Custom Board (no ICS) with MON08 serial port circuitry and additional auto-reset circuit built in. In this configuration, the ICS board is not used at all. The user must provide a serial port connection from the PC and all hardware configuration necessary to force the processor into MON08 mode upon reset. In addition, the user must include an extra circuit which allows the reset line of the processor to be driven low from the DTR line of the serial port connector (Pin 4 on a DB9). The following diagram shows the additional connection needed to reset from a DB9 serial connector.

Figure 13.14 Reset Connection



In this configuration, because the software does not directly control power to the processor, the user will be prompted to turn the processor's power supply on and off. Turning off the power supply is necessary in order to be able to pass the initial security mode check and access the flash on the processor. A simple reset is not enough; to pass the security check, you must first force the processor to encounter a POR (power-on reset) which requires the processor's voltage to dip below 0.1v. Once security has been passed, resetting the device should be facilitated by the above circuitry.

14

P&E Multilink/Cyclone PRO Connection

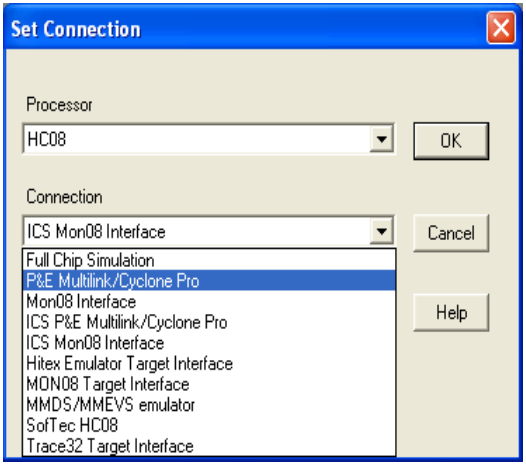
The HC08 P&E Multilink/Cyclone Pro Connection setting permits a connection to Class 5, 7 or 8 devices. Please see "Device Class Description" on page 379 for a definition of each interface class. HC08 P&E Multilink/Cyclone Pro connection mode allows the user to debug code, as the firmware is fully resident in the FLASH of the microprocessor. The operation of all modules fully reflects the actual operation of the on-board resources.

Connection Procedure

To select the P&E Multilink/Cyclone Pro connection:

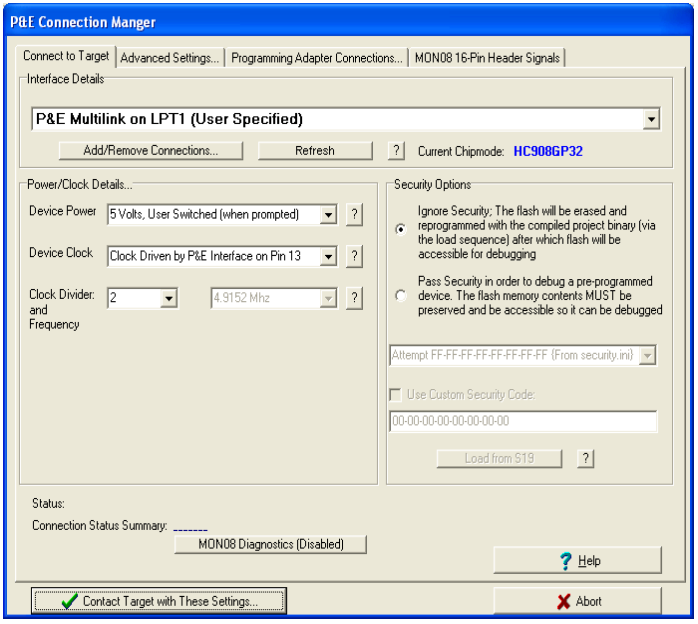
1. Choose the P&E Multilink/Cyclone Pro option from the Set Connection dialog box, as shown in Figure 14.1 on page 361.

Figure 14.1 Set Connections Dialog Box - P&E Multilink/Cyclone PRO Selected



2. Click on the OK button - The Connection Manager Window open with the Connect to Target Tab exposed, as shown in Figure 14.2 on page 362.

Figure 14.2 P&E Connection Manager Window - Connect to Target Tab


P&E Connection Manger
Connect to Target
Advanced Settings...
Programming Adapter Connections...
MON08 16-Pin Header Signals
Interface Details
P&E Multilink on LPT1 (User Specified)
Add/Remove Connections...
Refresh
Current Chipmode: HC908GP32
Power/Clock Details...
Device Power
5 Volts, User Switched (when prompted)
Device Clock
Clock Driven by P&E Interface on Pin 13
Clock Divider: and Frequency
2
4.9152 Mhz
Security Options
Ignore Security: The flash will be erased and reprogrammed with the compiled project binary (via the load sequence) after which flash will be accessible for debugging
Pass Security in order to debug a pre-programmed device. The flash memory contents MUST be preserved and be accessible so it can be debugged
Attempt FF-FF-FF-FF-FF-FF-FF-FF (From security.ini)
Use Custom Security Code:
00-00-00-00-00-00-00-00
Load from S19
Status:
Connection Status Summary:
MON08 Diagnostics (Disabled)
Help
Contact Target with These Settings...
Abort

3. The device power selection allows the user to specify whether the target is 2, 3, or 5 Volts, and whether this power is switched/generated by the P&E interface or if it is separately supplied to the target and under user control. If it is under user control, the software uses dialog boxes to ask the user to power the target up and down when necessary (similar to Class II-IV). See Figure 14.3 on page 362.

Figure 14.3 Device Power Selection List

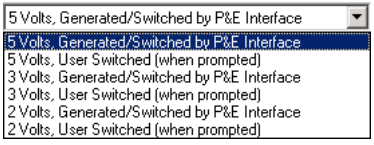
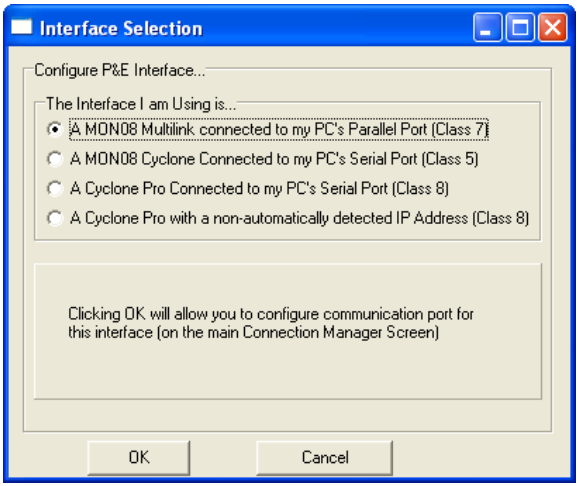

5 Volts, Generated/Switched by P&E Interface
5 Volts, Generated/Switched by P&E Interface
5 Volts, User Switched (when prompted)
3 Volts, Generated/Switched by P&E Interface
3 Volts, User Switched (when prompted)
2 Volts, Generated/Switched by P&E Interface
2 Volts, User Switched (when prompted)

Figure 14.4 HC08 Device Clock Selection Box

4. The device clock menu allows two options, as shown in Figure 14.4 on page 362.
   - P&E provides clock to target
   - The device has its own clock

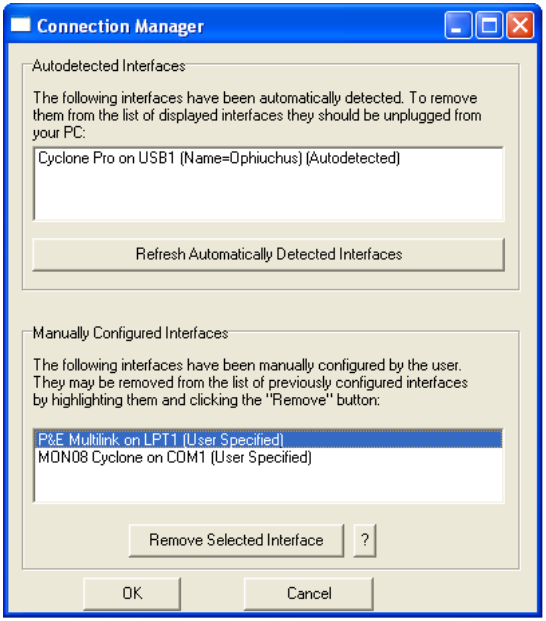
5. Click on “Refresh” which should display all P&E devices that are automatically detected, such as Cyclone Pro USB, Cyclone Pro Ethernet and USB-ML-MON08 Multilink. See Figure 14.2 on page 362.
6. To add a serial/parallel port P&E device such as Cyclone Pro Serial, MON08 Multilink and Cyclone Pro Ethernet (IP outside of subnet mask), proceed to the Interface Selection Window by pressing the “Add A Connection” button on this tab of the Connection Manager window. See Figure 14.5 on page 363. For more information about configuring Cyclone Pro for ethernet operation, please see the Cyclone Pro User's Manual.

Figure 14.5 Interface Selection Dialog Box


Interface Selection
Configure P&E Interface...
The Interface I am Using is...
A MON08 Multilink connected to my PC's Parallel Port (Class 7)
A MON08 Cyclone Connected to my PC's Serial Port (Class 5)
A Cyclone Pro Connected to my PC's Serial Port (Class 8)
A Cyclone Pro with a non-automatically detected IP Address (Class 8)
Clicking OK will allow you to configure communication port for this interface (on the main Connection Manager Screen)
OK
Cancel

7. To remove a manually configured interface, please click on the “Remove Connection” button button on this tab of the Connection Manager window and choose the interface to be deleted. See Figure 14.6 on page 364.

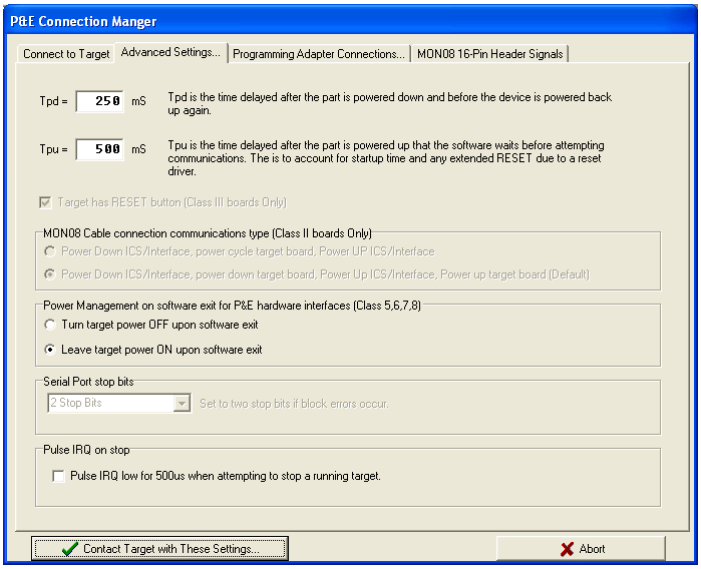
Figure 14.6 Connection Manager Dialog Box - Remove A Manually Configured Multilink/ Cyclone Pro Interface


Connection Manager
Autodetected Interfaces
The following interfaces have been automatically detected. To remove them from the list of displayed interfaces they should be unplugged from your PC:
Cyclone Pro on USB1 (Name=Ophiuchus) (Autodetected)
Refresh Automatically Detected Interfaces
Manually Configured Interfaces
The following interfaces have been manually configured by the user. They may be removed from the list of previously configured interfaces by highlighting them and clicking the "Remove" button:
P&E Multilink on LPT1 (User Specified)
MON08 Cyclone on COM1 (User Specified)
Remove Selected Interface
?
OK
Cancel

Advanced Settings Tab

The Advanced Settings tab of the P&E Connection Manager window allows the user to set specific protocol settings. The following is an explanation of each part of the advanced settings dialog.

Figure 14.7 Advanced Settings Tab


P&E Connection Manger
Connect to Target
Advanced Settings...
Programming Adapter Connections...
MON08 16-Pin Header Signals
Tpd =
250
mS
Tpd is the time delayed after the part is powered down and before the device is powered back up again.
Tpu =
500
mS
Tpu is the time delayed after the part is powered up that the software waits before attempting communications. The is to account for startup time and any extended RESET due to a reset driver.
Target has RESET button (Class III boards Only)
MON08 Cable connection communications type (Class II boards Only)
Power Down ICS/Interface, power cycle target board, Power UP ICS/Interface
Power Down ICS/Interface, power down target board, Power Up ICS/Interface, Power up target board (Default)
Power Management on software exit for P&E hardware interfaces (Class 5,6,7,8)
Turn target power OFF upon software exit
Leave target power ON upon software exit
Serial Port stop bits
2 Stop Bits
Set to two stop bits if block errors occur.
Pulse IRQ on stop
Pulse IRQ low for 500us when attempting to stop a running target.
Contact Target with These Settings...
Abort

Tpd And Tpu Timing Textboxes

Tpu and Tpd will set the power-up and power-down delay (respectively) that will be observed when power-cycling a target for entry into Monitor Mode. These settings are only valid for devices with automatically controlled power.

Whenever power is automatically switched off, or is manually requested to be switched off, the software waits for an amount of time equal to the Tpd delay time before proceeding to the connection protocol. This is because a board or power supply may have capacitance which holds the power up for a short time after the supply has been switched off, but the supply voltage must reach less than 0.1v before it is turned back on if a Power-On reset is to occur.

Whenever power is automatically switched on, or is manually requested to be switched on, the software waits for an amount of time equal to the Tpu delay time before attempting to contact the 68HC08 processor. This is to allow time not only for power to be fully available, but to wait until any reset driver has finally released the RESET line.

Target Has RESET Button Checkbox

With Class III boards, the software occasionally needs to get control of the target. On systems which are Class III boards with the monitor mode circuitry built-in (including RS-232 driver), there is no means to reset the target to gain control. If the board has a reset button, the software can use this to gain control of the target system. If this option is checked, the software will prompt the user to push the target reset button when a reset of the target system is desired. If the option is unchecked, the software will ask the user to power cycle the target system to achieve a reset.

MON08 Cable Connection Communications Type

This selection box is valid only for Class II hardware configurations using the MON08 cable. It allows the user to specify the sequence that the software uses to power up the ICS system. When the software tries to create a power-on reset condition, two events must occur:

1. Power of the target MCU must go below 0.1v. This means that the processor can not be receiving power from its power pins, nor can it have a significant voltage being driven on port pins or the IRQ line, as these will drive the MCU power back through these pins. It is crucial, therefore, to have the ICS and the Target both powered down at some point in time.
2. The processor MON08 configuration pins, including IRQ, must be properly driven when the target processor resets to drive it into monitor mode. If these pins are not set up properly before the processor powers up, the processor may start up in user mode.

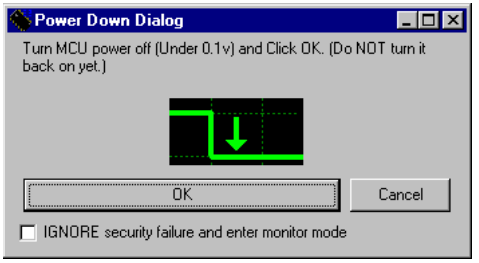
Power Up and Power Down Radio Buttons

Power Down ICS button asks the user to power down the board. Power Up ICS asks the user to power up the board

This is the default option and should work for most, if not all, ICS08/Target Board solutions. Refer to the manual addendum under startup for the settings for a specific ICS board. It requires the user go through two dialog stages, and requires more time than simply cycling the power.

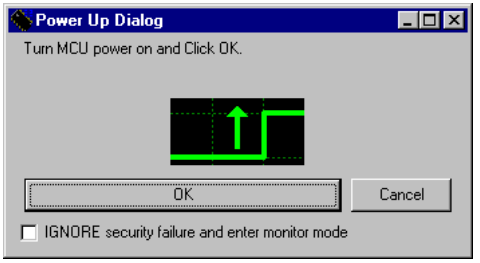
1. Software automatically powers down the ICS.
2. Software asks the user to power down the board as follows:

Figure 14.8 Power Down Dialog Box


Power Down Dialog
Turn MCU power off (Under 0.1v) and Click OK. (Do NOT turn it back on yet.)
OK
Cancel
IGNORE security failure and enter monitor mode

3. Software automatically powers up the ICS, which configures the processor's MON08 configuration pins.
4. Software asks the user to power up the board as follows:

Figure 14.9 Power Up Dialog Box



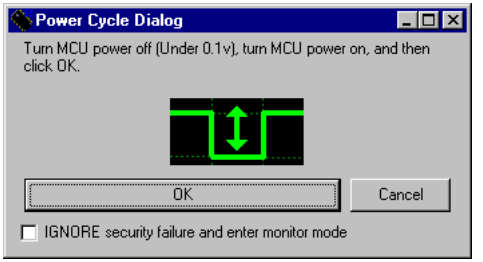
Cycle Power Down/Up Radio Button

These buttons ask the user to power cycle their board.

This option will work for many ICS boards as well, but relies on the fact that while the ICS is powered off, it will hold the target in reset until it is powered up itself and has configured the MON08 configuration pins. The sequence of events in this mode is:

1. Software automatically powers down the ICS.
2. Software asks the user to power cycle their board as follows:

Figure 14.10 Power Cycle Dialog Box


Power Cycle Dialog
Turn MCU power off (Under 0.1v), turn MCU power on, and then click OK.
OK
Cancel
IGNORE security failure and enter monitor mode

3. Software automatically powers up the ICS, which configures the processors MON08 configuration pins.

Serial Port Stop Bits

Serial Port Stop Bits allow users of Class 1-4 devices that are experiencing unreliable communication to increase the number of stop bits to 2.

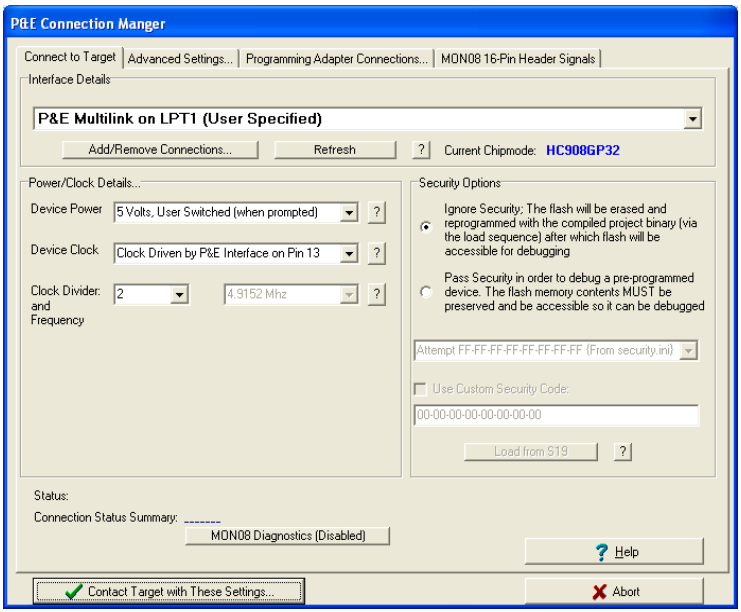
Pulse IRQ on Stop

The IRQ on Stop allows the users to execute a mechanism to stop a running HC08 microprocessor. For more information, please refer to the Stop a Running HC908 Target.pdf application note that is available from P&E Microcomputer Systems website: www.pemicro.com.

Target MCU Security Bytes

One of the steps that is necessary to properly bypass security is to provide the proper security code for the information that is programmed into the part. This holds true even when the part is blank. The security code consists of the 8 values which are currently stored in flash locations $FFF6 - $FFFD of the processor.

Figure 14.11 P&E Connection Manager Window - P&E Multilink on LPT1 Interface


P&E Connection Manger
Connect to Target
Advanced Settings...
Programming Adapter Connections...
MON08 16-Pin Header Signals
Interface Details
P&E Multilink on LPT1 (User Specified)
Add/Remove Connections...
Refresh
Current Chipmode: HC908GP32
Power/Clock Details...
Device Power
5 Volts, User Switched (when prompted)
Device Clock
Clock Driven by P&E Interface on Pin 13
Clock Divider: and Frequency
2
4.9152 Mhz
Security Options
Ignore Security: The flash will be erased and reprogrammed with the compiled project binary (via the load sequence) after which flash will be accessible for debugging
Pass Security in order to debug a pre-programmed device. The flash memory contents MUST be preserved and be accessible so it can be debugged
Attempt FF-FF-FF-FF-FF-FF-FF-FF (From security.ini)
Use Custom Security Code:
00-00-00-00-00-00-00-00
Load from S19
Status:
Connection Status Summary:
MON08 Diagnostics (Disabled)
Help
Contact Target with These Settings...
Abort

This window can be used to manually enter the proper security bytes via the USER setting, or to load the security bytes from the same .S19 file which was programmed. The bytes are loaded from an .S19 file by clicking the “Load from S19” button.

IGNORE Security Failure And Enter Monitor Mode

This checkbox can be used to cause the software to ignore a failure to properly pass the 68HC08 security check. If the checkbox is set, the software will attempt to establish monitor mode communications regardless of the security status. As long as the Baud and Port are correct, and the device has been properly powered, this will allow monitor mode entry. By ignoring the security check failure you may use monitor mode, but the ROM/Flash will not be accessible.

NOTE If a connection is not established for a reason other than security failure, the connection dialog will always appear.

STATUS Area

The status area of the P&E Connections Manager window consists of one status string following the “Status:” label, and seven items which list the state of the last attempt to connect to a target and pass security. The description for these items is as follows:

0 – ICS Hardware loopback detected:

Every ICS or board which supports MON08 has a serial loopback in hardware which, by connecting the transmit and receive lines, automatically echoes characters from the PC. A valid character transmitted from the PC should be echoed once by the loopback circuitry on the board and once by the monitor of the target processor itself. This status indicates whether or not the first echoed character from the hardware loopback was received when one of the security bytes was transmitted. If the status is ‘N’, which indicates that the character was not received, it is most likely due to one of the following reasons:

– Wrong Com Port specified.
– The baud rate specified was incorrect (probably too low).
– The ICS/Target is not connected.
– No Power to the ICS.

If this status bit responded with an ‘N’, you must correct this before analyzing the reset of the status bits.

1 – Device echoed some security bytes:

The monitor resident in a 68HC08 device automatically echoes every incoming character when it is in monitor mode. A valid character transmitted from the PC should be echoed once by the loopback circuitry on the board and once by the monitor of the target processor itself. This status indicates whether or not the second echoed character from the monitor response was received when one of the security bytes was transmitted. If the status is ‘N’, which indicates that the character was not received, or not received properly, it is most likely due to one of the following reasons:

– The baud rate specified was incorrect.
– The part did not start the monitor mode security check on reset. Signals to force monitor mode may be incorrect.
– No Power to the ICS.

If this status bit responded with an ‘N’, you must correct this before analyzing the reset of the status bits.

2 – Device echoed all security bytes:

In order to pass security, the software must send 8 security bytes to the processor. The processor should echo each of these eight bytes twice. If all 8 bytes did not get the proper two-byte echo, this flag will be ‘N’. Reasons for this include:

– The part did not start the monitor mode security check on reset. Signals to force monitor mode may be incorrect.

– The baud rate specified was incorrect.

– The processor was not reset properly. Check the “Target Hardware Type” and if you are connecting to a class II board, check the “MON08 cable communication connections type” in the “advanced settings” dialog.

3 – Device signaled monitor mode with a break:

Once the processor has properly received the 8 bytes from the PC software to complete its security check, it should transmit a break character to the PC signaling entry into monitor mode. This break should be sent regardless of whether the security check was successfully passed. If a break was not received from the processor, this flag will be ‘N’. Reasons for this include:

– The baud rate specified was incorrect.

– The processor was not reset properly. Check the “Target Hardware Type”. If you are connecting to a class II board, check the “MON08 cable communication connections type” in the “advanced settings” dialog.

4 – Device entered monitor mode:

Once the software has received, or failed to receive, a break from the processor, it attempts to communicate with the monitor running on the 68HC08 processor. It tries to read the monitor version number by issuing a monitor mode read. If the processor fails to respond properly to this command, this flag will be ‘N’.

5 – Reset was Power-On Reset:

If the device properly entered monitor mode (4), the software will read the reset status register (RSR). This read does not affect the security sequence, and occurs purely for diagnostic reasons. The reset status register indicates the conditions under which the processor underwent the last reset. For the software to pass the security check properly, it MUST first cause the processor to undergo a Power-On Reset. The software reads the reset status register to determine if the last reset was indeed caused by power-on. The result of the reset status register is indicated in parentheses after the flag value. If the highest bit is not set then the reset was not a power on reset, and the flag will indicate ‘N’. Reasons for this include:

– The processor did not power all the way down because power was being supplied to the processor through either the port pins, IRQ line, RESET line, or power pins.

– The voltage driven on the power pin of the processor did not go below 0.1 volts.

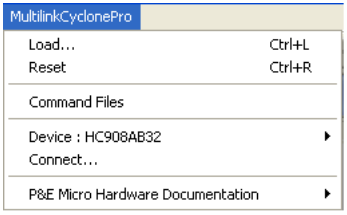
– The processor was not reset properly. Check the “Target Hardware Type”. If you are connecting to a class II board, check the “MON08 cable communication connections type” in the “advanced settings” dialog.

6 – ROM is accessible (un-secured):

If the device properly entered monitor mode (4), the software reads locations $FFF6-$FFFF to determine if the processor passes the security check. Memory locations which are invalid or protected read back from the device as $AD. If all bytes from $FFF6-$FFFF read a value of $AD, it is assumed the device is secure, and the flag value is an 'N'. If all flags 0-5 register a value of 'Y' and flag 6 register a value of 'N,' then the reset process has gone correctly except that the security code used to pass security was incorrect. Specify the correct security code and try again, or IGNORE the security failure and erase the device. Once you erase a secured device, you must exit the software and restart it in order to pass security.

Connection (MultilinkCyclonePro) Menu

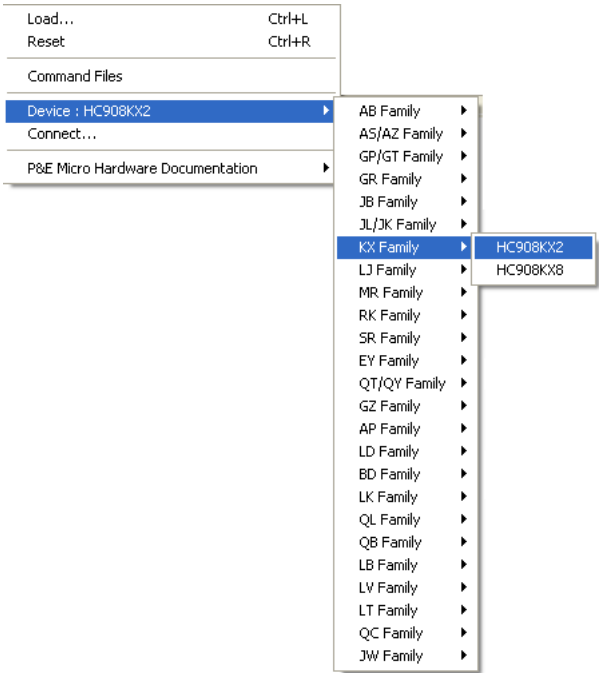
Figure 14.12 MultilinkCyclonePro Menu



Device Option

The "Device" option in the MultilinkCyclonePro menu allows the user to select the particular Freescale processor that they wish to use. When choosing Device from the MultilinkCyclonePro menu, extended menus will open which allow you to select the family (e.g. KX Family), and device type (e.g. HC908KX2) of the MCU that you are using.

Figure 14.13 HC08 Device Extended Menus



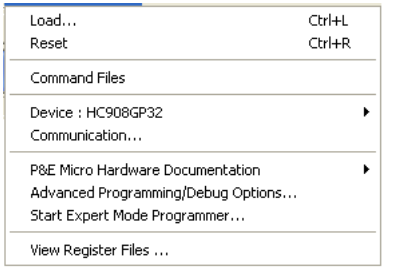
Connect Option

The “Connect” option initiates an attempt to communicate with the device chosen under the device section of the menu.

Active Mode Menu Options

When the microprocessor is connected, more Connection menu entries become available to the user.

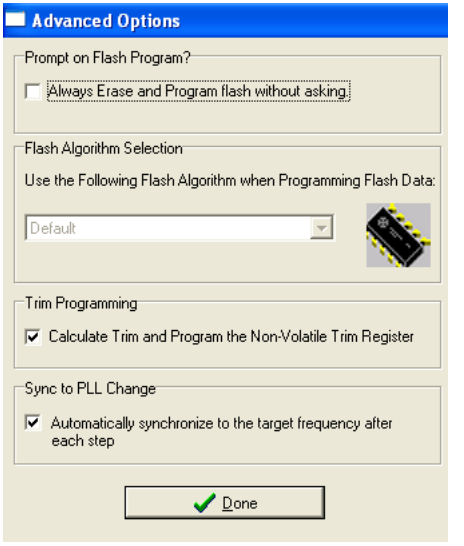
Figure 14.14 Additional Menu Options



Advanced Programming/Debug Options Option

The Start Advanced Programming/Debug Options menu entry take user to the Advanced Options dialog, where he can configure the software settings for the FLASH programming procedure.

Figure 14.15 Advanced Options Dialog Box


Advanced Options
Prompt on Flash Program?
Always Erase and Program flash without asking
Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
Default
Trim Programming
Calculate Trim and Program the Non-Volatile Trim Register
Sync to PLL Change
Automatically synchronize to the target frequency after each step
Done

Prompt on Flash Program Checkbox

Checking “Always Erase and Program flash without asking” lets the software transparently program the microprocessor.

Trim Programming Checkbox

The “Trim Programming” checkbox enables automatic calculation and programming of the trim value in a designated Non-Volatile memory location.

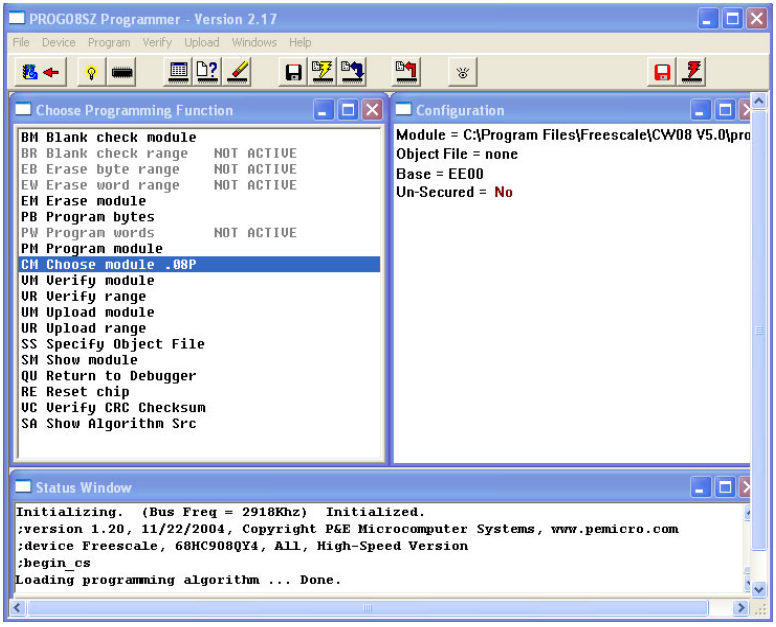
Sync to PLL Change Checkbox

“Sync to PLL Change” is required for the software/hardware connection to synchronize with the microprocessor during the flash erasing/programming procedure.

Start Expert Mode Programmer Option

“Start Expert Mode Programmer” grants the user access to P&E’s graphical flash programming utility, PROG08SZ. PROG08SZ lets an advanced user control the step-by-step execution of the flash erase/programming procedure. See Figure 14.16 on page 375. More information on how to use PROG08SZ can be found on P&E Microcomputer Systems website at: www.pemicro.com.

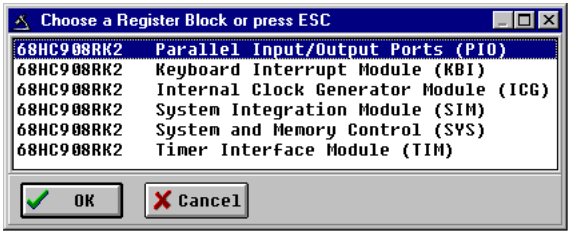
Figure 14.16 PROG08SZ Programmer WIndow


PROG08SZ Programmer - Version 2.17
File Device Program Verify Upload Windows Help
Choose Programming Function
BM Blank check module
BR Blank check range NOT ACTIVE
EB Erase byte range NOT ACTIVE
EW Erase word range NOT ACTIVE
EM Erase module
PB Program bytes
PW Program words NOT ACTIVE
PM Program module
CM Choose module .08P
VM Verify module
VR Verify range
UM Upload module
UR Upload range
SS Specify Object File
SM Show module
QU Return to Debugger
RE Reset chip
VC Verify CRC Checksum
SA Show Algorithm Src
Configuration
Module = C:\Program Files\Freescale\CW08 V5.0\pro
Object File = none
Base = EE00
Un-Secured = No
Status Window
Initializing. (Bus Freq = 2918Khz) Initialized.
;version 1.20, 11/22/2004, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Freescale, 68HC908QY4, All, High-Speed Version
;begin_cs
Loading programming algorithm ... Done.

View Register Files Option

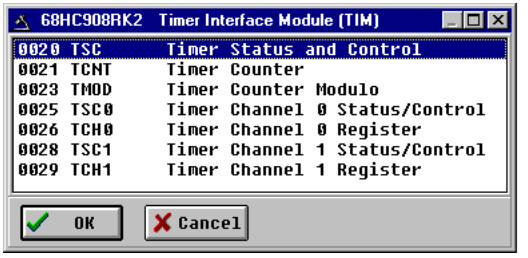
The “View Register Files...” menu selection also gives the user the option of running the register file viewer/editor. If register files are available for the device that you have chosen, the “Choose a Register Block” window (see Figure 14.17 on page 376) will be opened. You may also open it by entering the R command in the Command Window command line.

Figure 14.17 Choose A Register Block Window


Choose a Register Block or press ESC
68HC908RK2 Parallel Input/Output Ports (PIO)
68HC908RK2 Keyboard Interrupt Module (KBI)
68HC908RK2 Internal Clock Generator Module (ICG)
68HC908RK2 System Integration Module (SIM)
68HC908RK2 System and Memory Control (SYS)
68HC908RK2 Timer Interface Module (TIM)
OK
Cancel

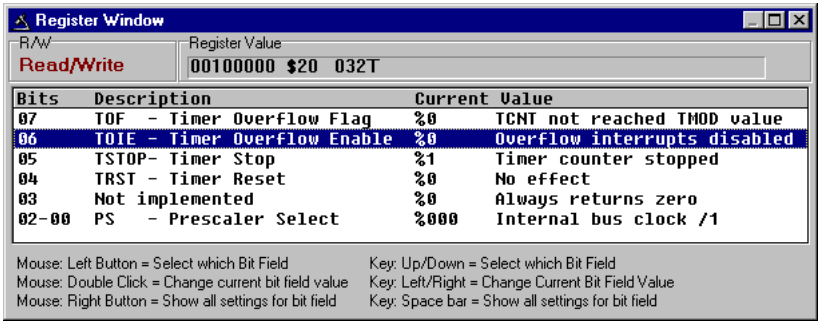
If register files have been installed on the host computer, selecting a block brings up the Register Block register listing (see Figure 14.18 on page 376), which shows a list of the associated registers, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.

Figure 14.18 Register Block Register Listing


68HC908RK2 Timer Interface Module (TIM)
0020 TSC Timer Status and Control
0021 TCNT Timer Counter
0023 TMOD Timer Counter Modulo
0025 TSC0 Timer Channel 0 Status/Control
0026 TCH0 Timer Channel 0 Register
0028 TSC1 Timer Channel 1 Status/Control
0029 TCH1 Timer Channel 1 Register
OK
Cancel

Selecting a file brings up the Register Window (see Figure 14.19 on page 377), which displays the values and significance for each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.

Figure 14.19 Register Window


Register Window
R/W
Read/Write
Register Value
00100000 $20 032T
Bits Description Current Value
07 TOF - Timer Overflow Flag %0 TCNT not reached TMOD value
06 TOIE - Timer Overflow Enable %0 Overflow interrupts disabled
05 TSTOP- Timer Stop %1 Timer counter stopped
04 TRST - Timer Reset %0 No effect
03 Not implemented %0 Always returns zero
02-00 PS - Prescaler Select %000 Internal bus clock /1
Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

Debugging Limitations

The following limitations are inherent in MON08 debugging and should be observed carefully.

1. Do not step a command that branches to itself.
2. Do not step an SWI (software interrupt) command.
3. The hardware breakpoint registers are reserved for use by the debugger. Attempting to use these registers for other purposes may not work.
4. Be careful about showing peripheral status and data registers in the memory or variables window. A refresh of the window will read these registers and may cause the clearing of flags.
5. The debug monitor built into CPU08 processors uses up to 13 bytes of the stack. Do not write to these addresses from (SP-13) to SP. To load a program into RAM, move the stack to the end of RAM.
6. If interrupts are turned on during stepping, the debugger will not step into the interrupt. Instead, it will execute the whole interrupt and stop on the command returned to after the interrupt.
7. Do not set hardware breakpoints within the monitor ROM area itself, or they will not function properly.

Debugging Tips

The following tips may prove useful:

1. Single stepping is allowed in both RAM or ROM.
2. The first breakpoint set is always a hardware breakpoint, and any additional breakpoints set are software breakpoints. To make sure that a hardware breakpoint is being set, make sure only one breakpoint is being used.
3. Hardware breakpoints will stop execution in ROM or RAM. Software breakpoints will stop execution only in RAM.
4. Experiment with the register interpreter. Use the “View Register Files...” option in the MultilinkCyclonePRO menu.
5. Executing an SWI instruction while running is functionally equivalent to hitting a breakpoint, except that execution stops at the instruction following the SWI.
6. A hardware breakpoint may be used to trap a data read/write to anywhere in the memory map. The debugger stops at the instruction after the one that accesses the data location.
7. When the target board is reset by the debugger, power to the microcontroller may be turned off for a short duration. Although much of RAM may look the same, some values may have changed.

Device Class Description

Class 5 Device

P&E's MON08 Cyclone Device connects to a PC via the serial port. The Cyclone-to-MCU connection occurs via a standard 16-pin MON08 ribbon cable. The MON08 Cyclone Device allows the user to auto-detect the baud rate as well as auto-cycle the power through the MCU system.

Class 7 Device

P&E MON08 Multilink and USB MON08 Multilink cables connect to the user's MCU system via a standard 16-pin MON08 ribbon cable. MON08 Multilink and USB MON08 Multilink allow one to auto-detect the baud rate as well as auto-cycle the power through the microprocessor system.

Class 8 Device

P&E's Cyclone Pro communicates with the PC through a Serial, Ethernet or USB port. The Cyclone Pro can be used to debug and program the firmware inside of Freescale HC08 microprocessors via a standard 16 pin MON08 ribbon cable. The Cyclone Pro can provide its own power and clock signals to the microprocessor, as long as proper signals are connected to the corresponding pins of the 16 pin MON08 header. In addition the Cyclone Pro can be used for programming and debugging the HC12/HCS12/HCS08 Freescale microprocessors via a standard 6 pin ribbon cable. To take advantage of this functionality, one should connect proper signals to the standard 6 pin Background Debug Module header.

15

ICS P&E Multilink/Cyclone PRO Connections

In-Circuit Simulation Mode is a P&E Microcomputer Systems mode of operation that is a hybrid between In-Circuit Debugging, and Full Chip simulation. P&E has combined the benefits of each of these modes, while minimizing their respective deficiencies.

In-Circuit Simulation, or ICS, mode simulates the CPU core instructions on the user's PC. However, Inputs/Outputs are read directly from the user's development device, and certain modules are run on the actual device. Any instructions that affect an I/O location-- for example, "lda PORTA" or "sta PORTA"-- would also use data from the real device. This allows for all the benefits of full chip simulation (Cycle by Cycle accuracy, unlimited breakpointing, and speed, to name just a few), while allowing the obvious benefit of garnering relevant signals from actual hardware.

In-Circuit simulation requires a user to have a connected MCU Device-- be it a development board, prototype hardware, or simply a device placed in a simple circuit on a breadboard. Once connection is established, the P&E Interface will load a .MON file to the microcontroller itself. This .MON file is the ICS-Kernel-- it constantly runs on the device, and interacts with the P&E software to provide information about changing I/O and any peripherals that are running on the device itself.

The following modules are run on the actual device when in ICS mode, and as such their Simulation commands become unnecessary, and therefore will not function:

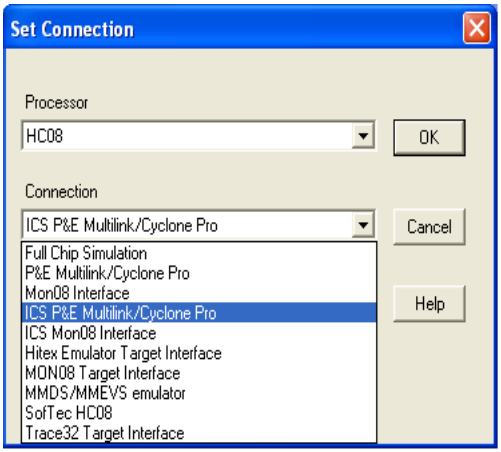
I/O ports, SCI, SPI, IRQ, ADC, LCD, KBI.

All other modules will function as fully simulated modules. For information about relevant simulation commands, please see the "HC08 Full Chip Simulation Mode" section above.

Connection Procedure

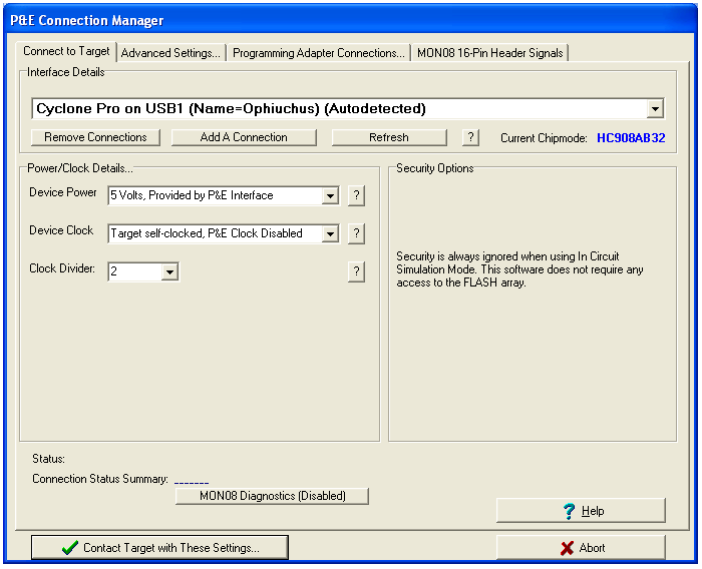
1. Choose the ICS P&E Multilink/Cyclone Pro option from the set connection dialog box, as shown in Figure 15.1 on page 382.

Figure 15.1 Set Connection Dialog Box



2. Click the OK button to bring up the P&E Connection Manager Window with its Connect to Target tab exposed.

Figure 15.2 P&E Connection Manager Window - Connect to Target Tab



3. The device power selection in the Power/Clock Details area allows the user to specify whether the target is 2, 3, or 5 Volts, and whether this power is switched/generated by the P&E interface or if it is separately supplied to the target and under user control. If it is under user control, the software will use dialog boxes to ask the user to power the target up and down when necessary (similar to Class II-IV). See Figure 15.3 on page 383.

Figure 15.3 Device Power Selection Box

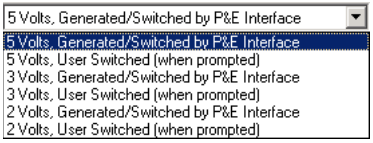
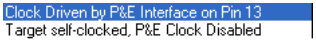

5 Volts, Generated/Switched by P&E Interface
5 Volts, Generated/Switched by P&E Interface
5 Volts, User Switched (when prompted)
3 Volts, Generated/Switched by P&E Interface
3 Volts, User Switched (when prompted)
2 Volts, Generated/Switched by P&E Interface
2 Volts, User Switched (when prompted)

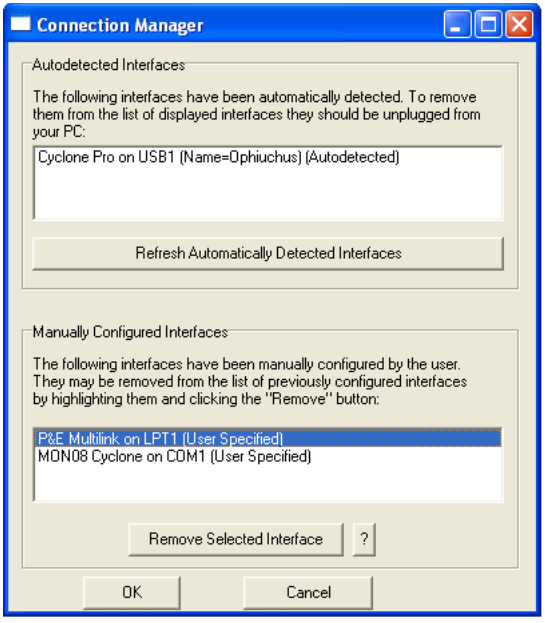
Figure 15.4 HC08 Device Clock Selection Box



4. The device clock menu allows two options. See Figure 15.4 on page 383.
   - P&E provides clock to target
   - The device has its own clock

5. Click on Refresh which should detect all P&E devices that are automatically detected, such as Cyclone Pro USB, Cyclone Pro Ethernet and USB ML MON08 Multilink. See Figure 15.2 on page 383.
6. To add a serial/parallel port P&E device such as Cyclone Pro Serial, MON08 Multilink and Cyclone Pro Ethernet (IP outside of subnet mask), proceed to Interface Selection Manager via pressing the Add A Connection button. For more information about configuring Cyclone Pro for ethernet operation, please see the Cyclone Pro User's Manual.
7. To remove a manually configured interface, please click on Remove Connection button and choose the interface to be deleted. See Figure 15.5 on page 384.

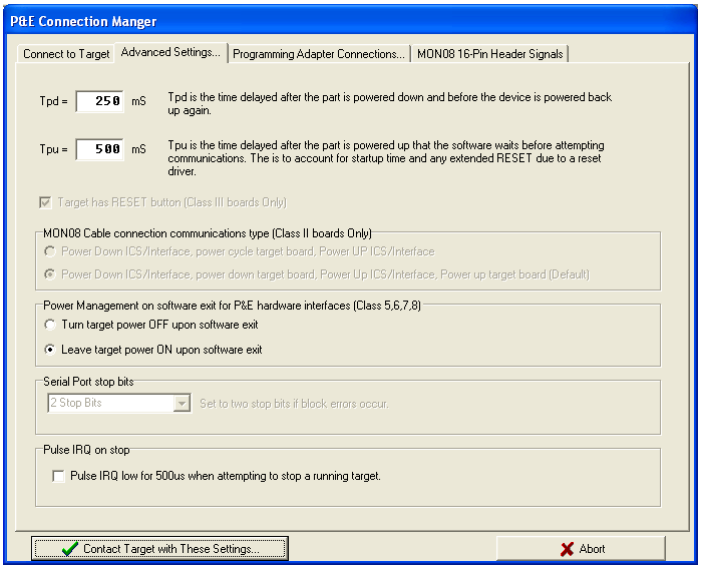
Figure 15.5 Remove A Manually Configured Multilink/Cyclone Pro Interface


Connection Manager
Autodetected Interfaces
The following interfaces have been automatically detected. To remove them from the list of displayed interfaces they should be unplugged from your PC:
Cyclone Pro on USB1 (Name=Ophiuchus) (Autodetected)
Refresh Automatically Detected Interfaces
Manually Configured Interfaces
The following interfaces have been manually configured by the user. They may be removed from the list of previously configured interfaces by highlighting them and clicking the "Remove" button:
P&E Multilink on LPT1 (User Specified)
MON08 Cyclone on COM1 (User Specified)
Remove Selected Interface
?
OK
Cancel

Advanced Settings Tab

The Advanced Settings tab of the P&E Connection Manager window allows the user to set specific protocol settings. The following is an explanation of each part of the Advanced Settings tab.

Figure 15.6 P&E Connection Manager Window - Advanced Settings Tab


P&E Connection Manger
Connect to Target
Advanced Settings...
Programming Adapter Connections...
MON08 16-Pin Header Signals
Tpd =
250
mS
Tpd is the time delayed after the part is powered down and before the device is powered back up again.
Tpu =
500
mS
Tpu is the time delayed after the part is powered up that the software waits before attempting communications. The is to account for startup time and any extended RESET due to a reset driver.
Target has RESET button (Class III boards Only)
MON08 Cable connection communications type (Class II boards Only)
Power Down ICS/Interface, power cycle target board, Power UP ICS/Interface
Power Down ICS/Interface, power down target board, Power Up ICS/Interface, Power up target board (Default)
Power Management on software exit for P&E hardware interfaces (Class 5,6,7,8)
Turn target power OFF upon software exit
Leave target power ON upon software exit
Serial Port stop bits
2 Stop Bits
Set to two stop bits if block errors occur.
Pulse IRQ on stop
Pulse IRQ low for 500us when attempting to stop a running target.
Contact Target with These Settings...
Abort

Tpd And Tpu Timing Textboxes

Tpu and Tpd will set the power-up and power-down delay (respectively) that will be observed when power-cycling a target for entry into Monitor Mode. These settings are only valid for devices with automatically controlled power.

Whenever power is automatically switched off, or is manually requested to be switched off, the software waits for an amount of time equal to the Tpd delay time before proceeding to the connection protocol. This is because a board or power supply may have capacitance which holds the power up for a short time after the supply has been switched off, but the supply voltage must reach less than 0.1v before it is turned back on if a Power-On reset is to occur.

Whenever power is automatically switched on, or is manually requested to be switched on, the software waits for an amount of time equal to the Tpu delay time before attempting to contact the 68HC08 processor. This is to allow time not only for power to be fully available, but to wait until any reset driver has finally released the RESET line.

Target Has RESET Button

In using Class III boards, the software occasionally needs to get control of the target. For a more detailed description of device classes, see "Device Class Description" on page 394. On systems which are Class III boards with the monitor mode circuitry built-in (including RS-232 driver), there is no means to reset the target to gain control. If the board has a reset button, the software can use this to gain control of the target system. If this option is checked, the software will prompt the user to push the target reset button when a reset of the target system is desired. If the option is unchecked, the software will ask the user to power cycle the target system to achieve a reset.

MON08 Cable Connection Communications Type

This selection box is valid only for Class II hardware configurations using the MON08 cable. For a more detailed description of device classes, see "Device Class Description" on page 394. It allows the user to specify the sequence that the software uses to power up the ICS system. When the software tries to create a power-on reset condition, two events must occur:

1. Power of the target MCU must go below 0.1v. This means that the processor can not be receiving power from its power pins, nor can it have a significant voltage being driven on port pins or the IRQ line, as these will drive the MCU power back through these pins. It is crucial, therefore, to have the ICS and the Target both powered down at some point in time.
2. The processor MON08 configuration pins, including IRQ, must be properly driven when the target processor resets to drive it into monitor mode. If these pins are not set up properly before the processor powers up, the processor may start up in user mode.

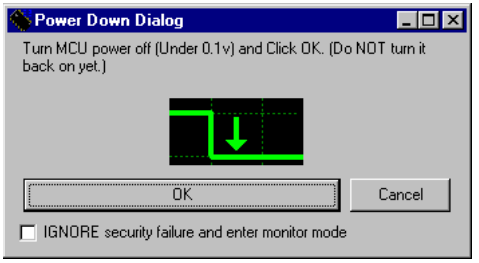
Power Up/Down Radio Buttons

Power Down ICS asks the user to power down the board. Power Up ICS assk the user to power up the board.

This is the default option and should work for most, if not all, ICS08/Target Board solutions. Refer to the manual addendum under startup for the settings for a specific ICS board. It requires the user go through two dialog stages, and requires more time than simply cycling the power.

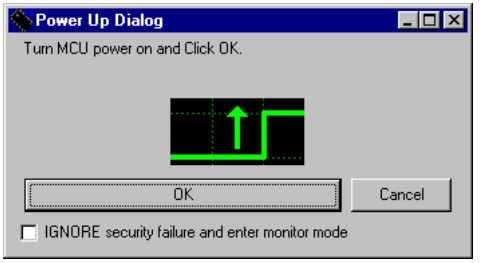
1. Software automatically powers down the ICS.
2. Software asks the user to power down the board as follows:

Figure 15.7 Power Down Dialog Box


Power Down Dialog
Turn MCU power off (Under 0.1v) and Click OK. (Do NOT turn it back on yet.)
OK
Cancel
IGNORE security failure and enter monitor mode

3. Software automatically powers up the ICS, which configures the processor's MON08 configuration pins.
4. Software asks the user to power up the board as follows:

Figure 15.8 Power Up Dialog Box



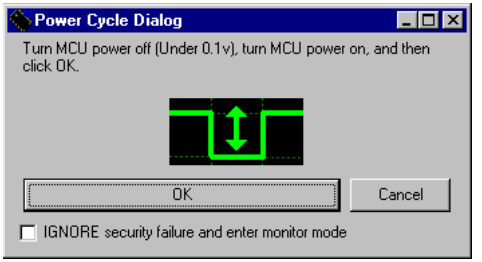
Cycle Power Up/Down Radio Buttons

Cycle Power ICS asks the user to power cycle the board.

This option will work for many ICS boards as well, but relies on the fact that while the ICS is powered off, it will hold the target in reset until it is powered up itself and has configured the MON08 configuration pins. The sequence of events in this mode is:

1. Software automatically powers down the ICS.
2. Software asks the user to power cycle their board as follows:

Figure 15.9 Power Cycle Dialog Box


Power Cycle Dialog
Turn MCU power off (Under 0.1v), turn MCU power on, and then click OK.
OK
Cancel
IGNORE security failure and enter monitor mode

3. Software automatically powers up the ICS, which configures the processors MON08 configuration pins.

Target MCU Security Bytes

In ICS mode, the user's code is simulated on the PC, and not run from the Non-Volatile memory of the device. Therefore, there is no need to pass security in this mode of operation.

STATUS Area

The status area of the P&E Connection Manager window consists of one status string following the “Status:” label, and seven items which list the state of the last attempt to connect to a target and pass security. The description for these items is as follows:

0 – ICS Hardware loopback detected:

Every ICS or board which supports MON08 has a serial loopback in hardware which, by connecting the transmit and receive lines, automatically echoes characters from the PC. A valid character transmitted from the PC should be echoed once by the loopback circuitry on the board and once by the monitor of the target processor itself. This status indicates whether or not the first echoed character from the hardware loopback was received when one of the security bytes was transmitted. If the status is ‘N’, which indicates that the character was not received, it is most likely due to one of the following reasons:

– Wrong Com Port specified.
– The baud rate specified was incorrect (probably too low).
– The ICS/Target is not connected.
– No Power to the ICS.

If this status bit responded with an ‘N’, you must correct this before analyzing the reset of the status bits.

1 – Device echoed some security bytes:

The monitor resident in a 68HC08 device automatically echoes every incoming character when it is in monitor mode. A valid character transmitted from the PC should be echoed once by the loopback circuitry on the board and once by the monitor of the target processor itself. This status indicates whether or not the second echoed character from the monitor response was received when one of the security bytes was transmitted. If the status is 'N', which indicates that the character was not received, or not received properly, it is most likely due to one of the following reasons:

– The baud rate specified was incorrect.

– The part did not start the monitor mode security check on reset. Signals to force monitor mode may be incorrect.

– No Power to the ICS.

If this status bit responded with an 'N', you must correct this before analyzing the reset of the status bits.

2 – Device echoed all security bytes:

In order to pass security, the software must send 8 security bytes to the processor. The processor should echo each of these eight bytes twice. If all 8 bytes did not get the proper two-byte echo, this flag will be 'N'. Reasons for this include:

– The part did not start the monitor mode security check on reset. Signals to force monitor mode may be incorrect.

– The baud rate specified was incorrect.

– The processor was not reset properly. Check the "Target Hardware Type" and if you are connecting to a class II board, check the "MON08 cable communication connections type" in the "advanced settings" dialog.

3 – Device signaled monitor mode with a break:

Once the processor has properly received the 8 bytes from the PC software to complete its security check, it should transmit a break character to the PC signaling entry into monitor mode. This break should be sent regardless of whether the security check was successfully passed. If a break was not received from the processor, this flag will be 'N'. Reasons for this include:

– The baud rate specified was incorrect.

– The processor was not reset properly. Check the "Target Hardware Type". If you are connecting to a class II board, check the "MON08 cable communication connections type" in the "advanced settings" dialog.

4 – Device entered monitor mode:

Once the software has received, or failed to receive, a break from the processor, it attempts to communicate with the monitor running on the 68HC08 processor. It

tries to read the monitor version number by issuing a monitor mode read. If the processor fails to respond properly to this command, this flag will be 'N'.

5 – Reset was Power-On Reset:

If the device properly entered monitor mode (4), the software will read the reset status register (RSR). This read does not affect the security sequence, and occurs purely for diagnostic reasons. The reset status register indicates the conditions under which the processor underwent the last reset. For the software to pass the security check properly, it MUST first cause the processor to undergo a Power-On Reset. The software reads the reset status register to determine if the last reset was indeed caused by power-on. The result of the reset status register is indicated in parentheses after the flag value. If the highest bit is not set then the reset was not a power on reset, and the flag will indicate 'N'. Reasons for this include:

– The processor did not power all the way down because power was being supplied to the processor through either the port pins, IRQ line, RESET line, or power pins.

– The voltage driven on the power pin of the processor did not go below 0.1 volts.

– The processor was not reset properly. Check the "Target Hardware Type". If you are connecting to a class II board, check the "MON08 cable communication connections type" in the "advanced settings" dialog.

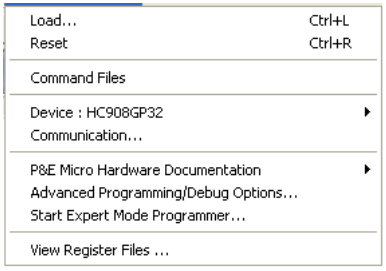
6 – ROM is accessible (un-secured):

If the device properly entered monitor mode (4), the software reads locations $FFF6-$FFFF to determine if the processor passes the security check. Memory locations which are invalid or protected read back from the device as $AD. If all bytes from $FFF6-$FFFF read a value of $AD, it is assumed the device is secure, and the flag value is an 'N'. If all flags 0-5 register a value of 'Y' and flag 6 register a value of 'N,' then the reset process has gone correctly except that the security code used to pass security was incorrect. Specify the correct security code and try again, or IGNORE the security failure and erase the device. Once you erase a secured device, you must exit the software and restart it in order to pass security.

Active Mode Menu Options

When the microprocessor is connected, more Connection menu options become available to the user.

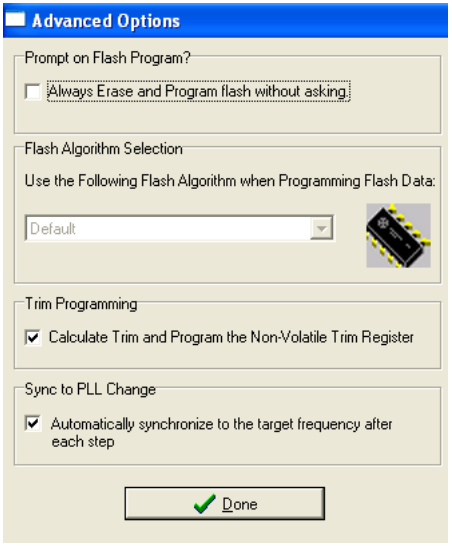
Figure 15.10 Additional Connection Menu Options



Advanced Programming/Debug Options Option

The Advanced Programming/Debug Options menu entry takes you to the Advanced Options dialog box, where you can configure the software settings for the FLASH programming procedure.

Figure 15.11 Advanced Options Dialog Box


Advanced Options
Prompt on Flash Program?
Always Erase and Program flash without asking
Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
Default
Trim Programming
Calculate Trim and Program the Non-Volatile Trim Register
Sync to PLL Change
Automatically synchronize to the target frequency after each step
Done

Prompt on Flash Program Checkbox

Checking “Always Erase and Program flash without asking” in this dialog box lets the software transparently program the microprocessor.

Trim Programming Checkbox

The “Trim Programming” checkbox enables automatic calculation and programming of the trim value in a designated Non-Volatile memory location.

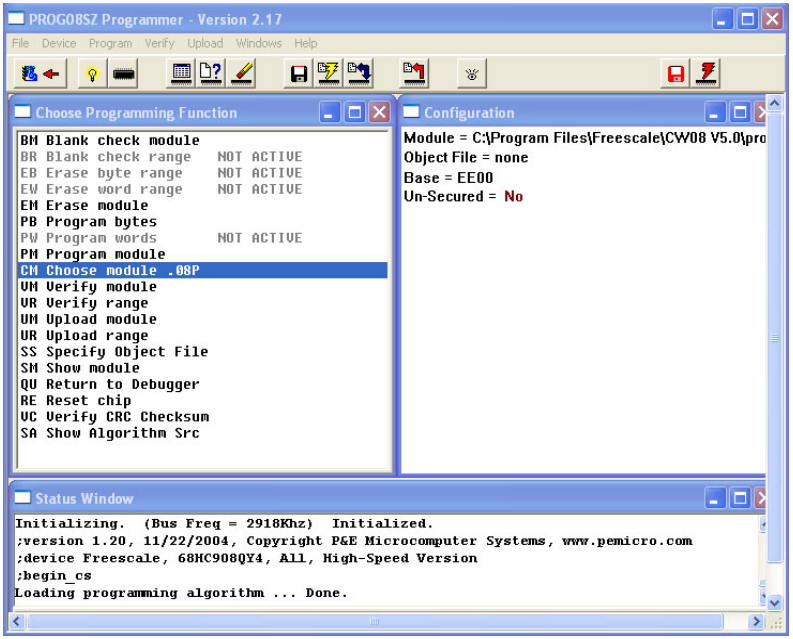
Sync to PLL Change Checkbox

“Sync to PLL Change” is required for the software/hardware connection to synchronize with the microprocessor during the flash erasing/programming procedure.

Start Expert Mode Programmer Option

“Start Expert Mode Programmer” grants the user access to P&E’s graphical flash programming utility, PROG08SZ. PROG08SZ lets an advanced user control the step-by-step execution of the flash erase/programming procedure. See Figure 15.12 on page 392. More information on how to use PROG08SZ can be found on P&E Microcomputer Systems website at: www.pemicro.com.

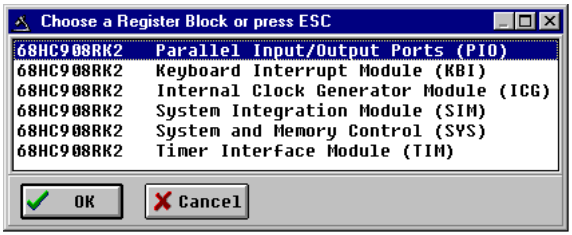
Figure 15.12 PROG08SZ Programmer Window


PROG08SZ Programmer - Version 2.17
File Device Program Verify Upload Windows Help
Choose Programming Function
BM Blank check module
BR Blank check range NOT ACTIVE
EB Erase byte range NOT ACTIVE
EW Erase word range NOT ACTIVE
EM Erase module
PB Program bytes
PW Program words NOT ACTIVE
PM Program module
CM Choose module .08P
VM Verify module
VR Verify range
UM Upload module
UR Upload range
SS Specify Object File
SM Show module
QU Return to Debugger
RE Reset chip
VC Verify CRC Checksum
SA Show Algorithm Src
Configuration
Module = C:\Program Files\Freescale\CW08 V5.0\pro
Object File = none
Base = EE00
Un-Secured = No
Status Window
Initializing. (Bus Freq = 2918Khz) Initialized.
;version 1.20, 11/22/2004, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Freescale, 68HC908QY4, All, High-Speed Version
;begin_cs
Loading programming algorithm ... Done.

View Register Files Option

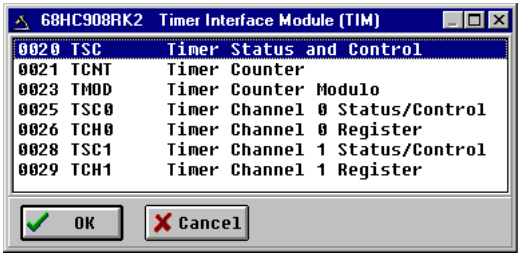
The “View Register Files...” menu selection also gives the user the option of running the register file viewer/editor. If register files are available for the device that you have chosen, the “Choose a Register Block” window (see Figure 15.13 on page 393) will be opened. You may also open it by entering the R command in the Command Window command line.

Figure 15.13 Choose A Register Block Window


Choose a Register Block or press ESC
68HC908RK2 Parallel Input/Output Ports (PIO)
68HC908RK2 Keyboard Interrupt Module (KBI)
68HC908RK2 Internal Clock Generator Module (ICG)
68HC908RK2 System Integration Module (SIM)
68HC908RK2 System and Memory Control (SYS)
68HC908RK2 Timer Interface Module (TIM)
OK
Cancel

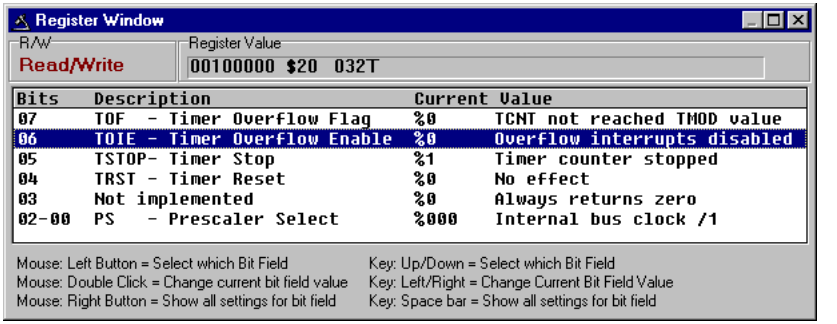
If register files have been installed on the host computer, selecting a block brings up the Register Block register listing (see Figure 15.14 on page 393), which shows a list of the associated registers, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.

Figure 15.14 Timer Interface Module Register Listing


68HC908RK2 Timer Interface Module (TIM)
0020 TSC Timer Status and Control
0021 TCNT Timer Counter
0023 TMOD Timer Counter Modulo
0025 TSC0 Timer Channel 0 Status/Control
0026 TCH0 Timer Channel 0 Register
0028 TSC1 Timer Channel 1 Status/Control
0029 TCH1 Timer Channel 1 Register
OK
Cancel

Selecting a file brings up the Register Window (see Figure 15.15 on page 394), which displays the values and significance for each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.

Figure 15.15 Register Window


Register Window
R/W
Read/Write
Register Value
00100000 $20 032T
Bits Description Current Value
07 TOF - Timer Overflow Flag %0 TCNT not reached TMOD value
06 TOIE - Timer Overflow Enable %0 Overflow interrupts disabled
05 TSTOP- Timer Stop %1 Timer counter stopped
04 TRST - Timer Reset %0 No effect
03 Not implemented %0 Always returns zero
02-00 PS - Prescaler Select %000 Internal bus clock /1
Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

Device Class Description

Class 5 Device

P&E MON08 Cyclone Device connects to PC via a serial port. The Cyclone to target connection takes place via a standard 16 pin MON08 ribbon cable. MON08 Cyclone Device allows one to auto detect the baud rate as well as auto cycle the power through the target.

Class 7 Device

P&E MON08 Multilink and USB MON08 Multilink cables connect to the target via a standard 16 pin MON08 ribbon cable. MON08 Multilink and USB MON08 Multilink allow one to auto detect the baud rate as well as auto cycle the power through the target.

Class 8 Device

P&E's Cyclone Pro communicates with the PC through a Serial, Ethernet or USB port.

One can use the Cyclone Pro to debug and program the firmware inside of Freescale

HC08 microprocessors via a standard 16 pin MON08 ribbon cable. The Cyclone Pro can provide its own power and clock signals to the target, as long as proper signals are connected to the corresponding pins of the 16 pin MON08 header.

In addition the Cyclone Pro can be used for programming and debugging the HC12/ HCS12/HCS08 Freescale microprocessors via a standard 6 pin ribbon cable. To take advantage of this functionality, one should connect proper signals to the standard 6 pin Background Debug Module header.

16

Softec HC08 Connection

This section guides you through the first steps toward debugging with CodeWarrior and the *SofTec HC08* connection. It does not replace all the additional documentation provided in this manual, but gives you a good starting point.

SofTec HC08 Technical Considerations

The 8/16 bits debugger (and then the CodeWarrior IDE) might be connected to HC08 hardware using the SofTec HC08.

When the debugger runs the **SofTec HC08** connection, it can communicate and debug **CPU08 (HC08)** core based hardware connected throught the SofTec in-circuit debugger/ programmer units, i.e:

SofTec Microsystems HC08 ISP Debuggers/Programmers (inDART Series) and Starter Kits (AK/SK/PK/ZK and newer Series).

Please refer to the "***inDART®-HC08 In-Circuit Debugger/Programmer for Motorola HC08 Family FLASH Devices User's Manual***" from SofTec for communication hardware requirements and SofTec product installation.

CodeWarrior and SofTec HC08 Connection

There are two seperate paths that may be followed to take the first steps toward debugging with Codewarrior and the SofTec inDART-HC08 connection. The differences between the two paths hinge on the starting point for the steps:

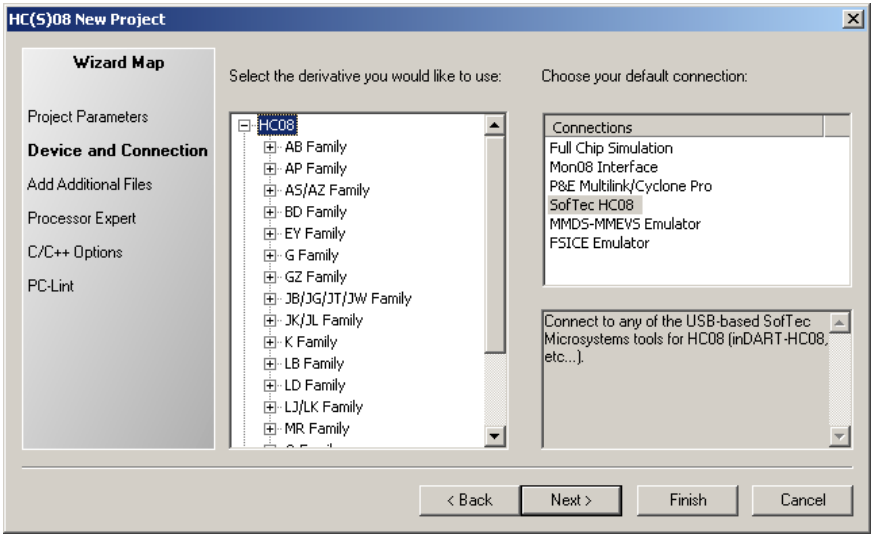
- Using the Stationary Wizard at the start of the project
- From within an existing project

Using the Stationery Wizard

To take the first steps toward debugging with CodeWarrior and theSofTec inDART-HC08 using the stationery Wizard:

1. Run the *CodeWarrior IDE* with the shortcut created in the program group.
2. Choose the menu **File > New** to create a new project from a stationery - the ***HC08 New Project Wizard first*** screen appears.
3. In the list box on the left of the screen, select the HC08 MCU you are targeting.
4. In the Connections list box, select ***SofTec HC08*** as the connection.

Figure 16.1 Wizard Connection Selection



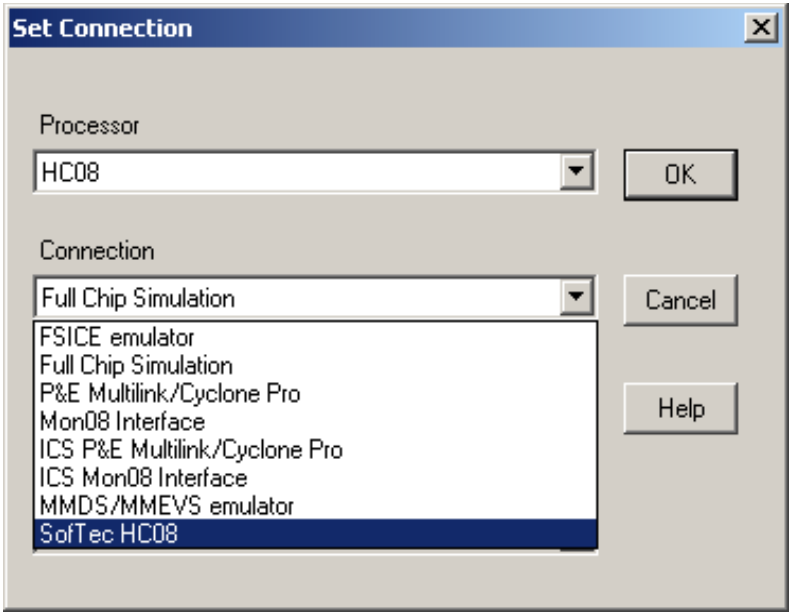
5. Finish the Wizard steps - the IDE opens.
6. In the IDE main window toolbar Project menu, choose **Project > Make.**
7. Now choose **Project > Debug** to start the debugger.

From Within an Existing Project

To take the first steps toward debugging with CodeWarrior and setting the SofTec HC08 connection from within an existing debugging project:

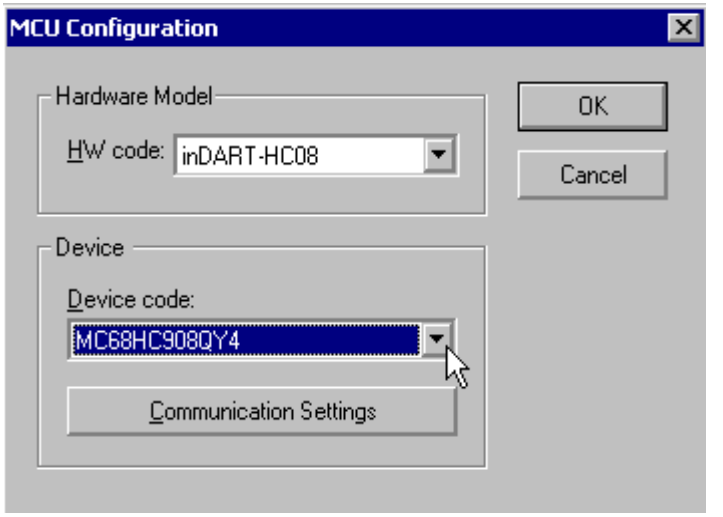
1. Run the *CodeWarrior IDE*.
2. Open the project.
3. Choose **Project > Debug** to start the debugger.
4. In the Debugger, choose **Component > Set Connection..** to select another target interface in the Set Connection dialog box.
5. Select ***HC08*** as Processor.
6. Select ***SofTec HC08*** as connection.

Figure 16.2 Set Connection Dialog Box - SofTec HC08 Selection



7. In the MCU Configuration dialog box, choose the correct target processor.

Figure 16.3 MCU Configuration Dialog Box

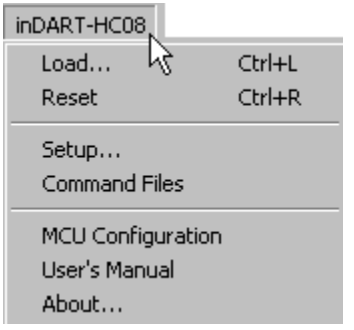


8. Press the OK button to start debugging.

inDart-HC08 Menu Options

Once the ***SofTec HC08*** connection is set, the connection menu entry in the debugger main toolbar is "***inDART-HC08***".

Figure 16.4 inDART-HC08 Menu Options



MCU Configuration Option

Select the **inDART-HC08 > MCU Configuration** *option* to display the MCU Configuration Dialog Box .

User's Manual Option

Select the **inDART-HC08 > User's Manual** option to open the ***"inDART®-HC08 In-Circuit Debugger/Programmer for Freescale HC08 Family FLASH Devices User's Manual"*** from SofTec.

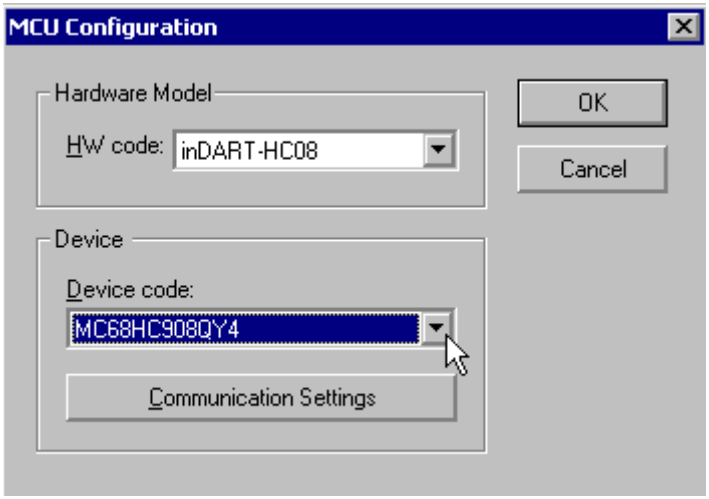
About Option

Select the **inDART-HC08 > About...** *option* to display the About Dialog Box .

MCU Configuration Dialog Box

The Hardware Model drop down list can be expanded to select another type of debug interface than the SofTec inDART-HC08. The *Device Code* drop down list can be expanded to select another HC08 derivative. Note that at this document release time, only the SofTec inDART-HC08 is available.

Figure 16.5 MCU Configuration Dialog Box



Pressing the *Communication Settings* button opens the Communication Settings Dialog Box.

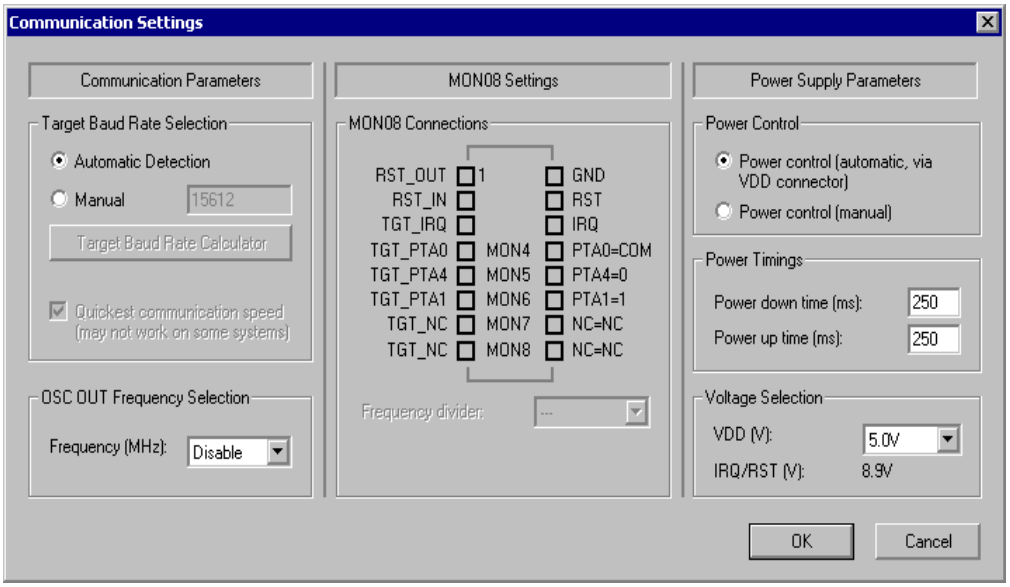
Communication Settings Dialog Box

Pressing the "*Communication Settings*" button in the MCU Configuration dialog box opens the Communication Settings dialog box, which allows you to fine-tune critical parameters needed for proper operation with the chosen target microcontroller.

The dialog box is divided into three sections: "*Communication Parameters*", "*MON08 Settings*" and "*Power Supply Parameters*". All of the parameters must be carefully set to ensure successful operation.

Refer to the ***"inDART®-HC08 In-Circuit Debugger/Programmer for Freescale HC08 Family FLASH Devices User's Manual"*** from SofTec for further details.

Figure 16.6 Communication Settings Dialog Box

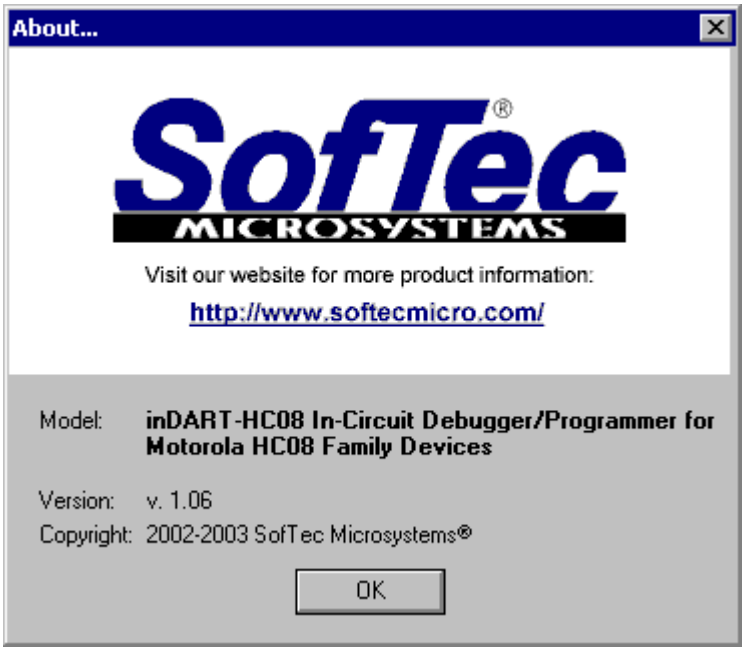


NOTE If your hardware supports stopping the application while running, an additional interrupt service routine is required for the IRQ vector. Please see ***Stop Command Handling*** section in ***"inDART®-HC08 In-Circuit Debugger/ Programmer for Motorola HC08 Family FLASH Devices User's Manual"*** from SofTec for further details.

About Dialog Box

This dialog box belongs to the SofTec GDI DLL and provides information about the ***inDART_HC08.dll*** release and version.

Figure 16.7 About Dialog Box



17

FSICEBASE Emulator

This manual is intended for developers, testers, application engineers, and anyone interested in using the Freescale In-Circuit Emulator Base (FSICEBASE) development system.

This manual contains information about how to use the FSICEBASE, tool that helps you develop applications for embedded systems based on a Freescale MC68HC08 microcontroller unit (MCU).

NOTE The Freescale In-Circuit Emulator Base (FSICEBASE) is similar to a system formerly produced by Motorola called the MMDS0508, a Motorola Modular Development System (MMDS). If you have worked with the MMDS0508, many of the features of the FSICEBASE will be familiar to you.

FSICEBASE Overview

This Overview section contains the following topics:

- System Requirements on page 403
- System Features on page 404
- System Components on page 405

The FSICEBASE works with CodeWarrior software, which is produced by Freescale, Inc. The CodeWarrior software provides an integrated development environment that includes an editor, assembler, and a user interface to the FSICEBASE system.

The environment allows you to perform source-level debugging. The CodeWarrior software also simplifies the process of managing and building a software project, and debugging code for an embedded MCU system. The benefit to you is reduced development time.

System Requirements

The FSICEBASE system requires a host computer with the following minimum specifications:

- Processor: 200 MHz Pentium® II processor or AMD-K6® class processor
- Operating System: Microsoft® Windows® 98/2000/XP

- RAM: 128 MB
- Hard drive space: Compact software installation: 232 MB
  Full software installation: 344 MB
- USB port or Ethernet port to connect host computer to the FSICEBASE

System Features

The Freescale In-Circuit Emulator Base (FSICEBASE) is a full-featured development system that provides in-circuit emulation. Features include:

- Real-time, non-intrusive, in-circuit emulation
- Real-time bus state analysis
- Meets ECC92 European electromagnetic compatibility standards
- Allows you to set four complex data or instruction breakpoints; a breakpoint can be qualified by an address, an address range, data, or externally connected logic clips.
- up to 128k real-time variables (any ROM ro RAM memory area)
- up to 128 Kbytes of emulation memory to accommodate the largest available ROM size of current HC08 MCU
- unlimited hardware instruction breakpoints over the 64-K memory map
- Built-in bus state analyzer:
  - 1.33Mb x 96 real-time trace buffer
  - Four hardware triggers for controlling real-time bus analysis and to provide breakpoints
  - Nine triggering modes
  - Display of real-time trace data as raw data, disassembled instructions, raw data and disassembled instructions, or assembly-language source code
  - As many as 1.33M pre- or post-trigger points
  - Trace buffer can be filled while single-stepping through user software
  - 32-bit time tag
  - Custom clock from 4100Hz to 40MHz in 5kHz steps, permitting wide time variance between analyzer events
  - 24 general-purpose logic clips, five of which can be used to trigger the bus state analyzer sequencer
- Four software-selectable internally generated oscillator clock sources
- Command and response logging to disk files
- Assembly-language source-level debugging

- On-screen, context-sensitive help via pop-up menus and windows
- Emulation that allows multiple types of reset

System Components

The FSICEBASE system includes the basic components that you need to connect to an emulation module (EM). You can also separately purchase additional components that can enhance debugging and emulation.

Basic Components

The Freescale In-Circuit Emulator Base includes the following components

- Base station

  The connectors on the top of the box let you connect an emulation module (EM).
- Cables, connectors, and adapters:
  - crossover ethernet cable (connects directly to an Ethernet Network Interface Card (NIC) on a PC)
  - straight-through ethernet cable (connects to a hub or switch)
  - Universal Serial Bus (USB) cable
  - MON08 debug port (built into base station)
  - external universal power supply and power supply cable
- System software

  CodeWarrior software, featuring an editor, assembler, and assembly source level debugger
- Documentation:
  - Freescale In-Circuit Emulator Base User Manual (this manual)
  - CodeWarrior User's Manual
  - Freescale In-Circuit Emulator Base Quick Start
  - Online Help and PDFs
- Two logic clip cable assemblies: twisted-pair cables that connect the station module to your target system. You can also use the cable assembly to connect a test fixture, a clock, an oscillator, or any other circuitry that you might use to perform analysis. One end of each cable assembly has a molded connector, which fits into the FSICEBASE. Leads at the other end of each cable terminate in female probe tips. Ball clips come with the cables.

Additional Components

You can purchase other components to enhance your development efforts. You can purchase these components separately:

- Host computer

  You must provide an IBM-compatible personal computer to run the development software.

- An emulation module (EM)

  An emulation module (EM) is a printed circuit board that emulates the features of a specific set of microcontroller units (MCUs). An EM completes the functionality of the FSICEBASE for a particular MCU or MCU family. The FSICEBASE works with a variety of EMs. You can purchase EMs separately from the FSICEBASE.

  The two DIN connectors on the bottom of the EM fit into connectors on the top of the FSICEBASE box. The target provides power and signal targets.

  Connection to your target system is then made through a separately purchased target cable and target head adapter that attaches to a target connector located on the top of the EM board.

- Optional target cable

  You can separately purchase a target cable that is part of a cable assembly, which is used to connect a target system to the FSICEBASE.

- Optional target head adapter

  You can separately purchase a target head adapter that is part of a cable assembly, which is used to connect a target system to the FSICEBASE.

- Optional Bus State Analyzer (BSA) cables

  The base station contains ports for three BSA pods. You can purchase BSA cables in addition to those supplied with the FSICEBASE system.

Setting Up The FSICEBASE System

The Freescale In-Circuit Emulator Base (FSICEBASE) development system includes cables and software. You need to connect the cables and install the software in order to use the FSICEBASE. This section contains the following topics to help you set up the system:

- Setting Up The Hardware on page 407
- Establishing Communication on page 408

Setting Up The Hardware

This section explains how to connect a host computer to the Freescale In-Circuit Emulator Base (FSICEBASE). There are three ways to connect a host computer to the FSICEBASE:

- Directly from the USB port of a host computer to the FSICEBASE USB port
- Directly from the ethernet port of a host computer to the FSICEBASE ethernet port
- From the host computer, through a Local Area Network (LAN), to the FSICEBASE ethernet port

A. Connect the FSICEBASE to a host computer in one of the three ways described in the following paragraphs.

1. If you are using an ethernet connection to connect your host computer to the FSICEBASE through a Local Area Network (LAN):
   a. Connect host computer to LAN
   b. Connect FSICEBASE to LAN
   c. Make sure power supply is not connected to board
   d. Connect one end of ethernet cable to ethernet port of FSICEBASE (make sure to use the straight-through ethernet cable when connecting to LAN)
   e. Connect other end of ethernet cable to Local Area Network (LAN)

NOTE To complete the connection through a LAN, you will need to obtain the IP address, subnet mask, and default gateway information from your network administrator. You will use this information in a later step.

2. If you are using an ethernet connection to connect your host computer directly to the FSICEBASE (not through a LAN):
   a. Make sure power supply is not connected to board
   b. Connect one end of ethernet cable to ethernet port of FSICEBASE (make sure to use the cross-over ethernet cable when connecting directly to a Network Interlace Card (NIC))
   c. Connect other end of USB cable to host computer

NOTE The host computer (PC) must have an assigned IP address and subnet mask that matches the FSICEBASE.

3. If you are using a USB connection to connect your host computer directly to the FSICEBASE:
   a. Make sure power supply is not connected to board
   b. Connect U-shaped end of USB cable to FSICEBASE

c. Connect other end of USB cable to host computer

B. Once you have connected the host computer to the FSCIBASE, connect Power supply to FSICEBASE

a. Connect round end of 5-volt power cord to barrel connector on FSICEBASE

b. Plug power supply into surge-protected strip

c. Connect surge-protected strip to AC outlet

C. Switch FSICEBASE Power switch to ON

LED lights after the base station finished boot sequence.

There are three status LEDs on the box: busy, ready, and error. The FSICEBASE base station takes about 5 seconds to boot. After powering the unit, you must wait for the ready LED before attempting to connect.

The FSICEBASE is now ready to accept communication with a host computer. You will need to install the CodeWarrior software, create a project, and start the debugger to establish communication between your host computer and the FSICEBASE.

Establishing Communication

The Freescale In-Circuit Emulator Base (FSICEBASE) allows you to connect to a host computer in two ways:

- Through an ethernet port
- Through a USB port

Communication Through Ethernet Port

If you use an Ethernet connection to establish communication between your host computer and the FSICEBASE through a LAN, you need to do three things:

- Have network administrator assign IP address on LAN to the FSICEBASE
- Set IP address on FSICEBASE
- Specify IP address in the debugger

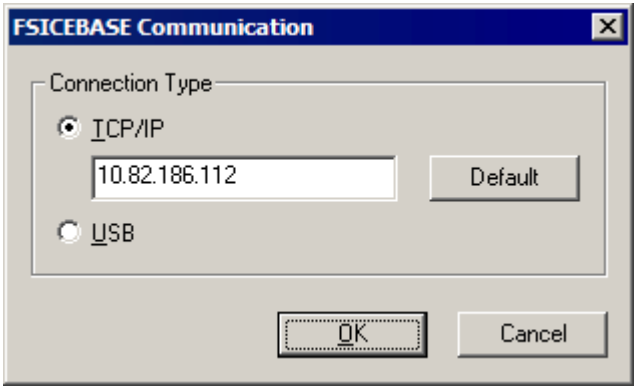
The following procedures explains how to do these things step by step.

To establish communication through a LAN:

1. Set up hardware as explained in “Setting Up The Hardware” .
2. Obtain the IP address that your network administrator assigned to the FSICEBASE
3. Start the debugger as explained in “Starting the Debugger” .)

4. Make sure connection is FSICEBASE
   a. From debugger main menu, select **Component**
   b. Select **Set Connection** — Set Connection dialog box appears
   c. Select **FSICEBASE emulator** from connection drop-down box
   d. Click **OK** — the debugger adds the FSICEBASE-HC08 menu to the main menu bar
5. From debugger main menu, select **FSICEBASE-HC08**
6. Select **Communication**— Communication dialog box appears (Figure 17.1 on page 409)

Figure 17.1 Communication Dialog Box



7. Select **TCP/IP**
8. In the text box, type the IP Address that your network administrator assigned to the FSICEBASE

NOTE For more information on the IP address of the FSICEBASE, see "Assigning An IP Address To FSICEBASE".

9. Click **OK**

   The debugger connects to the FSICEBASE through the ethernet port.

Communication Through USB Port

If you use a USB connection to establish communication between your host computer and the FSICEBASE:

1. Set up hardware as explained in "Setting Up The Hardware" .
2. Start the debugger as explained in "Starting the Debugger" .)

NOTE If you have started the debugger from your project previously, when you select Debug in the IDE, the Debugger attempts to connect to the FSICEBASE with the last known settings. If the Debugger connects, you do not need to perform the following steps.

3. Make sure connection is FSICEBASE
   a. From debugger main menu, select **Component**
   b. Select **Set Connection** — Set Connection dialog box appears
   c. Select **FSICEBASE emulator** from Connection drop-down box
   d. Click **OK**
4. From debugger main menu, select **FSICEBASE-HC08**
5. Select **Communication**— Communication dialog box appears (Figure 17.1 on page 409)
6. Select **USB**
7. Click **OK**

   The debugger connects to the FSICEBASE through the USB port.

Setting Up The System

In order to use the Freescale In-Circuit Emulator Base (FSICEBASE), you need to make sure that the system is configured properly. Configuring the FSICE system includes:

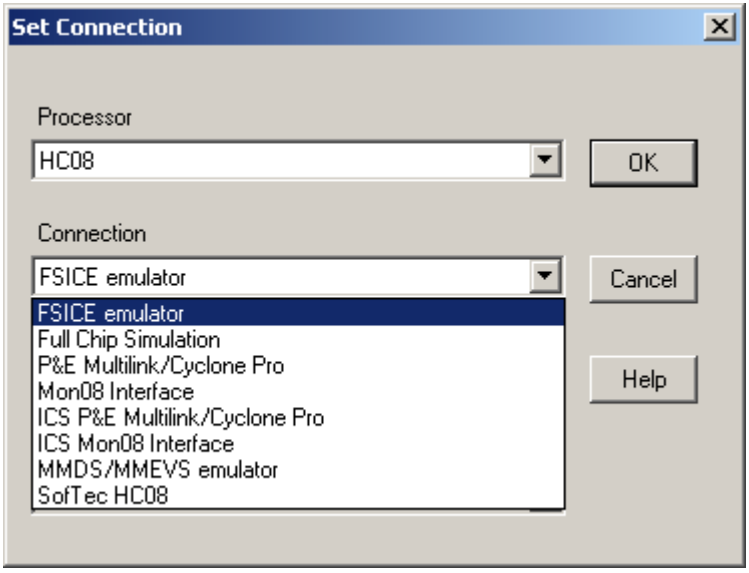
- Specifying A Connection on page 411
- Specifying Communication Information on page 412
- Assigning An IP Address To FSICEBASE on page 413
- Specifying a Memory Map on page 416
- Specifying the Clock Speed on page 419

Specifying A Connection

To specify the connection, including the specific derivative:

1. Start the debugger — the **True-time Simulator & Real-time Debugger** window appears.
2. From debugger main menu, select **Component**
3. Select **Set Connection** from the Component menu — Set Connection dialog box opens

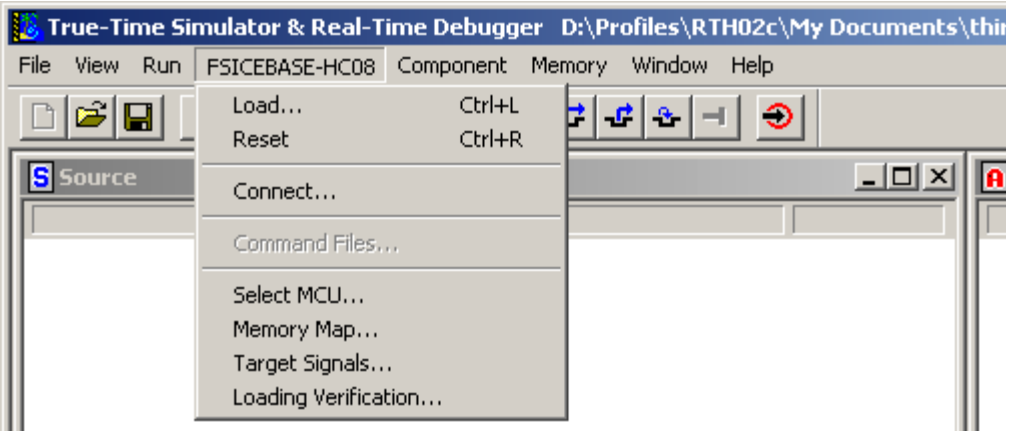
Figure 17.2 Set Connection Dialog Box



4. Select appropriate processor from **Processor** drop-down menu
5. Select appropriate **connection**. To specify the FSICEBASE as the connection, select **FSICEBASE emulator**.
6. Click **OK** — The debugger configures itself to work with the connection that you specified.

Notice that the main menu of the debugger reflects your selection. The menu item between the **Run** menu and the **Component** menu shows the name of the connection that you selected. For example, if you selected FSICE emulator as the connection, the main menu contains a menu item labeled **FSCICEBASE-HC08** as shown in Figure 17.3 on page 412. The FSICEBASE-HC08 dropdown menu is also shown.

Figure 17.3 Debugger Main Window Toolbar



Specifying Communication Information

When you start the debugger from a the CodeWarrior IDE, the debugger automatically prompts you to specify communication information. However, if necessary, you can change the communication information directly from the debugger.

To specify communication information:

1. Start the debugger — the **True-time Simulator & Real-time Debugger** window appears.
2. From debugger main menu, select **FSCICEBASE-HC08**

   The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the connection. For more information on specifying a connection see, "Specifying A Connection" .
3. Select **Communication** — The FSICE Communication dialog box opens (Figure 17.1 on page 409)
4. Specify communication information
   a. If you use an ethernet connection to connect your host computer to the FSICEBASE through a LAN:
      - Select **TCP/IP,** and
      - Type the IP address of the FSICEBASE in the text box.

NOTE The network administrator of your Local Area Network (LAN) needs to assign the IP address of the FSICEBASE on the network. You can use the default IP address, and give this address to your network administrator. Or, your network

administrator might choose to create a different IP address. If the network administrator chooses the IP address, you need to assign the IP address to the FSICEBASE. For more information see, "Assigning An IP Address To FSICEBASE".

b. If you use an ethernet connection to connect your host computer directly to the FSICEBASE (not through a LAN):
   - In the Communication dialog box, select **TCP/IP,** and
   - In the text box of the Communication dialog box, type the IP address of the FSICEBASE

NOTE Make sure that the cable between the host computer and the FSICEBASE is a cross-over ethernet cable.

NOTE Make sure that the host computer uses a static IP address. (The FSICEBASE does not assign an IP address to the host computer.) If you use the default IP address of the FSICEBASE (192.168.0.1), we recommend that you assign the following IP address to the host computer: 192.168.0.2.

NOTE Make sure that the host computer and FSICEBASE both use the same subnet mask.

c. If you use a USB cable to connect your host computer directly to the FSICEBASE station, select **USB**,

5. Click **OK**

   The debugger attempts to connect to the FSICEBASE. An information box shows the progress. You can click **Cancel** in the information box if you do not want to immediately connect to the FSICEBASE.

The debugger saves the communication information that you specified. It uses the communication information the next time that it connects to the FSICEBASE.

Assigning An IP Address To FSICEBASE

The FSICEBASE ships from the factory with the following internal default IP address:

```
192.168.0.1
```

Depending on how you connect the host computer to the FSICEBASE, you might need to change the IP address of the FSICEBASE. The CodeWarrior software includes a utility that allows you to assign a different IP address to the FSICEBASE.

To assign an IP address to the FSICEBASE:

1. Use a USB cable to connect the host computer to the FSICEBASE
2. Start the FSICEBASE Configuration Utility
   a. From Windows desktop, click **Start** menu
   b. Select **Run**
   c. Browse to the following executable file:

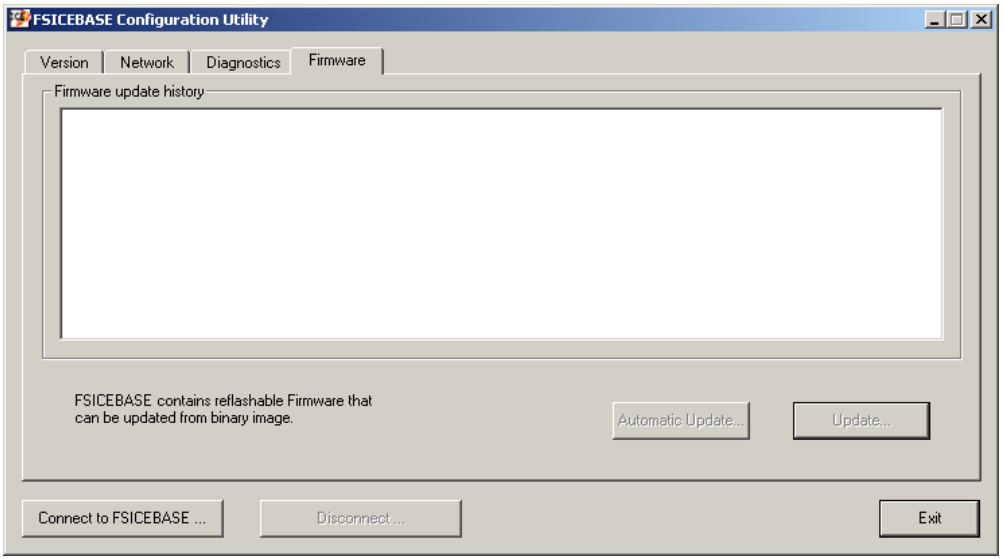
      `installation_directory\prog\GDI\FSICEBASE\setup.exe`

      The `installation_directory` is the directory where you installed the CodeWarrior software. The default installation directory is

      `C:\Program Files\Freescale\CW08 v5.x`

   d. Click **OK** — The **FSICEBASE Configuration Utility** starts (Figure 17.4 on page 414)

Figure 17.4 FSICEBASE Configuration Utility Window



3. Click **Connect to FSICEBASE** button — **FSICEBASE Communication** dialog box appears, as shown in Figure 17.1 on page 409.
4. Select **USB**

NOTE You can also use the default IP address to connect through TCP/IP.

5. Click **OK** — the FSICEBASE Configuration Utility connects to the FSICEBASE

6. Click **Network** tab of FSICEBASE Configuration Utility
7. In **Address** text box, type the IP address that you want to assign to the FSICEBASE

NOTE All hosts on a network must have a unique IP address. If you are connecting the FSICEBASE to a Local Area Network (LAN), consult with your network administrator to obtain a valid IP address.

8. From **Mask** combo box, select the subnet mask that you want to assign to the FSICEBASE

NOTE All the hosts in a sub-network must have the same subnet mask. For that reason, if you connect the host computer directly to the FSICEBASE (not through a LAN), you must ensure that the host computer uses the same subnet mask as the FSICEBASE.

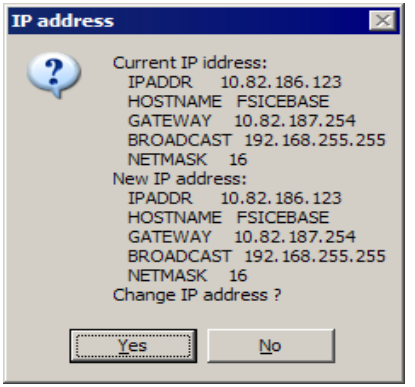
9. If applicable, in **Default Gateway** text box, type the IP address of the gateway that the FSICEBASE should use to connect to a network
10. If applicable, in **Broadcast Address** text box, type the IP address that you want the FSICEBASE to use as the broadcast address on the network

NOTE The broadcast IP address is the last IP address in the range of IP addresses on a network. The broadcast address is reserved by the network to allow a single host to make an announcement to all hosts on the network. Consult your network administrator for more information.

11. Click **Change** button

    The FSICEBASE Configuration Utility displays the IP Address Change Confirmation dialog box (Figure 17.5 on page 416) with the information that you specified. If information is wrong, click **No** to return to the Network tab and correct the information.

Figure 17.5 IP Address Change Confirmation Dialog Box



12. Click **Yes**

The FSICEBASE Configuration Utility assigns the new IP information to the FSICEBASE.

Specifying a Memory Map

Different MCU designs require different memory map configurations of the FSICEBASE system.

A personality file defines memory maps for particular MCUs. The personality file defines the memory map of each MCU supported by an emulator module (EM). Personality files ship with the separately purchased EMs. Refer to the appropriate EM user's manual to determine the personality files used by a particular EM module.

If an EM is connected to the FSICEBASE, the CodeWarrior software automatically loads the default personality file that corresponds to the EM. If the CodeWarrior software does not find an appropriate personality file, the debugger displays an error message when it tries to connect to the FSICEBASE.

After the debugger has loaded a memory map, you can view the memory map and modify it.

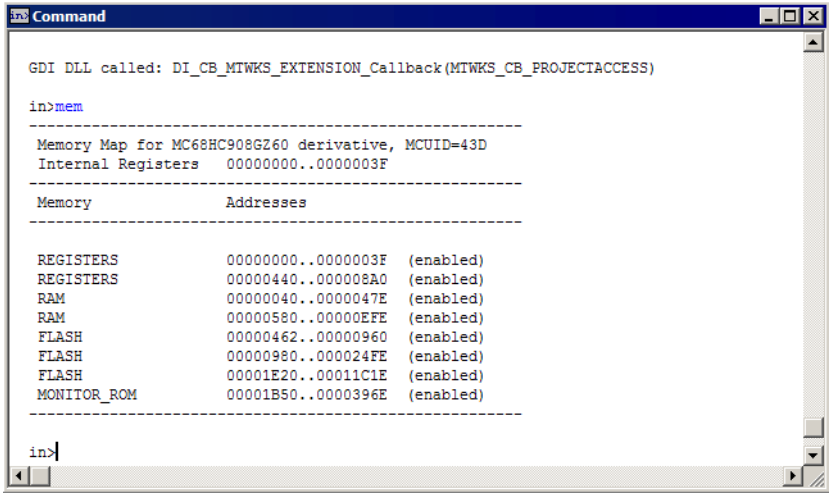
To use the Command line to view the current memory map:

1. From debugger main menu, select **Window** > **Command** to view the Command window. If you do not see the Command window:
   a. From the debugger main menu select **Component** > **Open**
   b. Select **Command**
   c. Click **OK** — the debugger opens a new Command window.

2. Click on command line (place insertion point on command line)
3. Type `MEM`

   Command window (Figure 17.6 on page 417) displays memory map information: a representation of the current system memory map, and the lower and upper boundaries of the internal module that contains the MCU registers.

Figure 17.6 Command Window in the Debugger

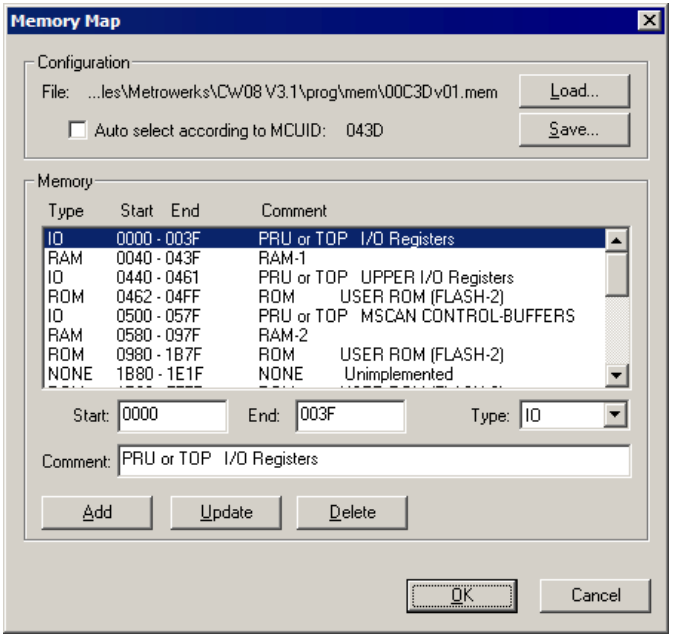


To Modify a Memory Map:

1. Start the debugger — the **True-time Simulator & Real-time Debugger** window appears.
2. From debugger main menu, select **FSCICEBASE-HC08**

   The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the connection. For more information on specifying a connection see, “Specifying A Connection” .
3. Select **Memory Map** — Memory Map dialog box opens (Figure 17.7 on page 418)

Figure 17.7 Memory Map Dialog Box



4. Specify memory map information
   a. From **Memory** list box, select portion of map to change
   b. In **Start** text box, type new start address of range desired
   c. In **End** text box, type new end address of range desired
   d. Select **Type** of memory the new range should represent
   e. In **Comment** text box, type new description of range if appropriate
   f. Click **Update** button to update highlighted range, or **Add** button to add a new range (be careful not to overlap ranges)
5. To delete an existing range:
   a. From Memory list box, select portion of map to delete
   b. Click **Delete**
6. To Save the definitions of the memory map that you specified:
   a. Click **Save** — Save Memory Configuration dialog box appears
   b. In File Name text box, type name you want to give the memory map file (`.mem` file)
   c. Click **Save** — debugger saves `.mem` file, which you can use (load) later

7. Click **OK** — The debugger loads the new memory map information. The Command window of the debugger shows confirmation message.

Specifying the Clock Speed

The FSICEBASE platform board can supply an oscillator clock source for the MCU's OSC1 input. Note that many emulator modules (EMs) require a specific jumper configuration so that this clock source can be used. Refer to the specific EM user's manual for EM clock source information.

The FSICEBASE has seven clock frequencies available: six internally generated clock frequencies (32 MHz, 16 MHz, 8 MHz, 4 MHz, 2 MHz, and 1 MHz) and an external clock source. You can also define a custom internal clock speed.

If you use an external clock source, you need to use a logic clip to connect the clock to the FSICEBASE. You must use logic clip A. Use the white wire to connect to the external clock.

To specify the clock speed:

1. Start the debugger — the **True-time Simulator & Real-time Debugger** window appears.

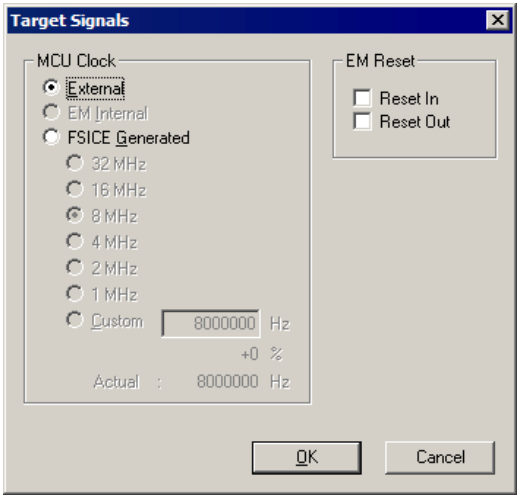
   For more information see "Starting the Debugger" .

2. From debugger main menu, select **FSCICEBASE-HC08**

   The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the connection. For more information on specifying a connection see, "Specifying A Connection" .

3. Select **Target Signals** — Target Signals dialog box opens (Figure 17.8 on page 420)

Figure 17.8 Target Signals Dialog Box



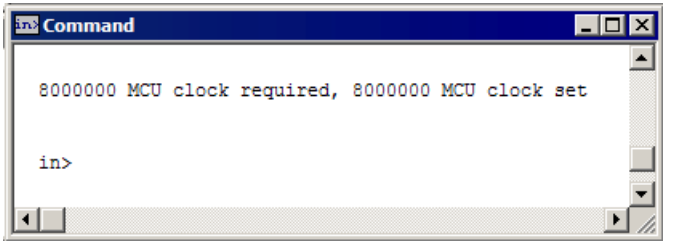
4. Specify clock source. From **MCU Clock** section of dialog box, select whether the clock is connected externally, on a connected emulator module (EM), or FSICE Generated.
5. Specify clock speed if internally generated
   a. If you selected **FSICE Generated**, select the clock speed to be emulated
   b. If you selected **Custom**, type clock speed in **Custom** text box

NOTE If you specify a custom clock speed, be aware that the FSICEBASE can provide clock speeds from 4100Hz to 40MHz in steps of 5kHz. The FSICEBASE uses a clock sythesis chip to generate the clock speed. This method is not as accurate as a crystal: only within about 0.75% absolute frequency with about 5% jitter. If you choose a clock value (either from one of the radio buttons or by typing it in) that is an integer divisor of 32MHz or 9.8304MHz, you will get an accurate crystal-sourced clock.

6. Click **OK**

The debugger instructs the FSICEBASE to use new clock information. The Command window (Figure 17.9 on page 421) of the debugger shows confirmation message.

Figure 17.9 Confirmation Message in Command Window



Emulation System Reset

The debugger allows you to reset the emulation MCU and set the PC register to the contents of the reset vector.

To reset the FSICEBASE:

1. If the FSICEBASE is connected to an emulator module (EM), specify the type of reset available to the EM.
   a. From debugger main menu, select **FSCICEBASE-HC08**

      The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the connection. For more information on specifying a connection see, "Specifying A Connection" .
   b. Select **Target Signals** — Target Signals dialog box opens (Figure 17.8 on page 420)
   c. If you want to allow a reset signal coming from the target system (through the target cable), check the **Reset IN** checkbox.

      Some EMs include a hardware jumper that governs target resets. Make sure to configure jumpers as necessary to use the **Reset IN** option. For more information, refer to your EM's documentation.
   d. If you want to allow a reset signal to be sent to the target system (through the target cable), check the **Reset Out** checkbox.

NOTE If you check both **Reset IN** and **Reset Out** the internal resets of the emulator system will not be sent to the target system.

   e. Click **OK**
2. From debugger main menu, select **FSCICEBASE-HC08**

3. Select **Reset**

   The debugger sends a reset signal to the FSICEBASE.

Setting Up Logic Cables And Connectors

The diagram below shows the pin numbering for both pod A and pod B logic cable connectors of the station module. Table 17.1 on page 422 shows the pinout information of the logic clips. You can use the logic clips are used to capture data in the bus state analyzer. (Pin 9 of both pods provides connection to an external ground.) In addition, the pod connectors are used as external clock inputs for the emulator clock and bus state analyzer timetag. The table also provides color code information for each pod. The external clock inputs are through pin 17 of each pod. Pod A pin 17 is the external clock input for the emulator. To use this source, make the desired clock connection to the white probe tip and use the OSC command to select an external source.

Pod B pin 17 is the external timetag input for the bus state analyzer. To use this source, make the desired clock connection to the white probe tip and use the TIMETAG command to select an external time tag source for the analyzer.

Table 17.1 Pod and Logic Cable Pin Assignments

Pod Pin	Pod A Signal	Pod B Signal	Probe Color
1	LC0	LC8	Brown (BRN)
2	GND	GND	
3	LC1	LC9	Red (RED)
4	GND	GND	
5	LC2	LC10	Orange (ORG)
6	GND	GND	
7	LC3	LC11	Yellow (YEL
8	GND	GND	
9	LC4	LC12	Green (GRN)
10	GND	GND	
11	LC5	LC13	Blue (BLU)
12	GND	GND	

Table 17.1 Pod and Logic Cable Pin Assignments (*continued*)

Pod Pin	Pod A Signal	Pod B Signal	Probe Color
13	LC6	LC14	Purple (PUR)
14	GND	GND	
15	LC7	LC15	Gray (GRY)
16	GND	GND	
17	EXT_OSC	TT_OSC	White
18	GND	GND	
19	GND	GND	Black
20	GND	GND	

Bus State Analyzer (BSA)

The bus state analyzer (BSA) shows the logical state of the target MCU bus. The BSA takes a snapshot of the MCU bus. It also captures the signals from the logic clips of Pods A, B, and C of the FSICBASE (24 lines in total). This capturing of data enables you to determine what is occurring in a system without actually disturbing the system.

At the end of each MCU clock cycle, the BSA takes a snapshot of the logical states of the target MCU bus. The analyzer stores the snapshots in the trace buffer, according to its mode. (This action is known as storing cycles.)

NOTE This analyzer is a bus state analyzer. It does not show signal hold or setup times.

To start using the BSA, you need to define patterns of logical states as events (or terms). You also need to specify the analyzer mode: continuous, counted, or any of five sequential modes. This determines which cycles the analyzer stores.

Data collection (cycle storage) begins when you arm the analyzer and start program execution. Data collection continues until execution stops, through a specified number of events, or through a defined sequence of events.

Using BSA

To use the bus state analyzer (BSA) to produce useful data that you can view and analyze, you must:

1. Define events (terms).
2. Arm the BSA.

Defining Events

You define an event by specifying a combination of criteria. You can define the criteria to be particular values in certain addresses, read or write access on an instruction or on data, extended address access, or signals sent through one of the five logic clips that you can connect to Pod A of the FSICEBASE.

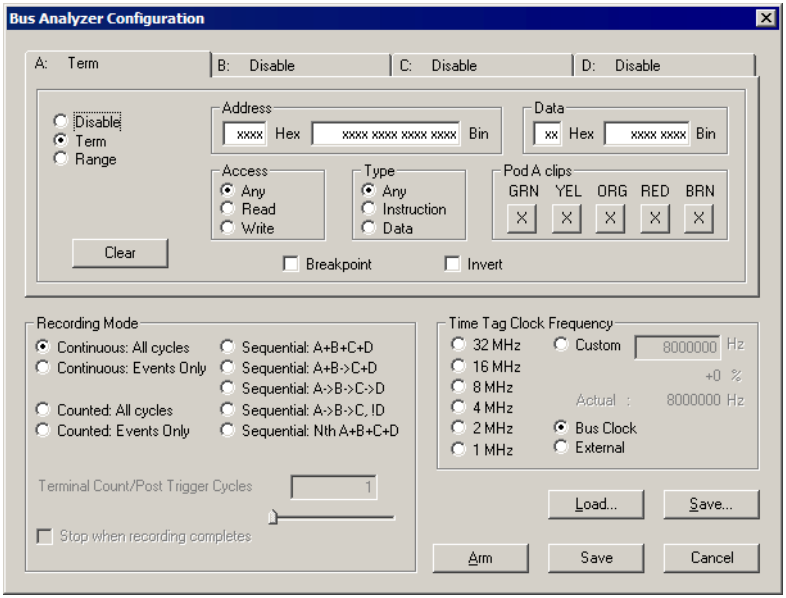
The Bus State Analyzer uses the criteria that you specify to create an event, and labels the event A, B, C, or D. When the BSA determines that the criteria of a certain event has been met, depending on the triggering mode, it records the data that is in the bus of the MCU at that particular clock cycle. It also records the data that is in the lines of Pods A, B, and C. You can control the way that the BSA records this information by specifying a recording mode.

To define an event:

1. Start the debugger
2. Load the program to debug
   a. If you launched the debugger from a project in the CodeWarrior IDE, the debugger automatically loads the program (`.abs` or `.elf` file)
   b. If the debugger has not loaded the program that you want to debug:
      - From the debugger main menu, select **File** > **Load Application**
      - Specify the location of the executable program file (`.abs` or `.s19` file)
      - Click **Open** — debugger loads the application you specified
3. From debugger main menu, select **FSCICEBASE-HC08**

   The FSICEBASE-HC08 menu is between the Run menu and the Component menu. If you do not see the FSICEBASE-HC08 menu, you need to specify the connection. For more information on specifying a connection see, "Specifying A Connection" .
4. Select **Bus Analyzer Configuration** — Bus Analyzer Configuration dialog box opens (Figure 17.10 on page 425)

Figure 17.10 Bus Analyzer Configuration Dialog Box



5. Select **Term** or **Range**

   A range consists of two 32-bit values. Range does not refer to a range of addresses. If you define an event as a range, the BSA triggers every time the input falls between the range starting term (the first 32-bit value) and the range ending term (the second 32-bit value).

6. In **Address** area, specify the address(es) that the BSA monitors
7. In **Data** area, specify the data that the BSA monitors
8. In **Access** area, specify the type of access that the BSA should monitor
9. In Type area, specify whether the BSA should record only if encountering data, instructions or any kind of value at the specified address
10. Specify Pod A signals (logic clips attach to pins of Pod A) that the BSA should monitor for this event

NOTE You can use five of the pod A logic clips to define an event. The other signals of Pod A, and the signals of Pods B and C cannot be used to define an event. The Bus Analyzer Configuration dialog box shows the five clips that you can use to trigger an event. The choice of these five signals is hard-set in the FSICEBASE; you cannot choose other signals to be used as event criteria. However, the BSA does capture data from all 24 lines of Pods A, B, and C.

11. If you want the term to also acts as a breakpoint, check the **Breakpoint** checkbox
12. Specify the Recording Mode
    - For information about the recording mode, see "Recording Modes" on page 426.
    - If you check the **Stop when recording completes** checkbox, the debugger stops program execution when bus state analyzer recording is done.
13. Specify the Time Tag Clock Frequency
14. Click **Save** to apply the event information to the current debug session and close the dialog box. The BSA uses the terms when you arm the BSA.
15. Click **Save...** to save the event information to a file
16. Click **Arm** to ready the BSA to collect data.

    The BSA does not start collecting data until execution begins.The debugger indicates that the BSA is armed by showing the word Armed in the status bar.

Recording Modes

When you define an event, you can specify the recording mode that the Bus State Analyzer uses to collect data. This section explains how the different modes work.

Continuous: All Cycles

After execution begins, the trace buffer begins storing data from the first cycle. This continues until execution arrives at a breakpoint, or until you halt execution.

Continuous: Events Only

After execution begins, the trace buffer begins storing data when data matches an event definition. This continues until execution arrives at a breakpoint, or until you halt execution

Counted: All Cycles

After execution begins, the trace buffer begins storing data after the specified number of cycles from first cycle. (A breakpoint can stop storage before the analyzer stores the specified number of cycles, as can halting execution.)

Counted: Events Only

After execution begins, the trace buffer begins storing data that match an event definition for the specified number of cycles. (A breakpoint can stop storage before the analyzer stores the specified number of cycles, as can halting execution.)

A+B+C+D

After execution begins, the trace buffer begins storing data from the first cycle run. This continues through the occurrence of event A, B, C, or D (whichever is enabled); data storage ends after the specified number of post-trigger cycles.

A+B -> C+D

After execution begins, the trace buffer begins storing data from the first cycle. This continues through the occurrence of two events: A or B, followed by C or D. Data storage ends after the specified number of post-trigger cycles.

If you select this mode, you must enable event A, event B, or both. You must enable event C, event D, or both. Otherwise, the bus state analyzer cannot be triggered.

A -> B -> C !D

After execution begins, the trace buffer begins storing data from all cycles. This continues through the occurrence of three events, A, B, and C, in order, if event D does not occur. (If D occurs, the sequencer starts again looking for event A.) Data storage ends after the specified number of post-trigger cycles.

If you select this mode, you must enable events A, B, and C. Otherwise, the bus state analyzer never can be triggered. If you disable event D, you convert this mode to a simple, three-event sequence.

A -> B -> C -> D

After execution begins, the trace buffer begins storing data from all cycles. This continues through the occurrence of four events, A, B, C, and D, in order. Data storage ends after the specified number of post trigger cycles.

If you select this mode, you must enable all four events A, B, C, then D. Otherwise, the bus state analyzer cannot be triggered.

Nth Event: A+B+C+D

After execution begins, the trace buffer begins storing data from N occurrences of cycles that match the definitions of events A, B, C, or D (whichever are enabled). Then the bus state analyzer captures the next 4096 cycles.

By selecting the terminal post trigger count, the user can control the number of cycles that is stored. This can be used to speed uploading of the BSA data if only a small portion of data is needed.

NOTE The terminal count or post trigger cycles are valid only for counted or sequential modes. For a counted mode, this field specifies the number of cycles

to be stored. For a sequential mode, this field specifies the number of cycles to be stored after the trigger sequence occurs.

Time Tag Clock Frequency

An optional part of analyzer setup is specifying the frequency and source of the time tag clock. This clock provides a time reference value in each frame of the trace buffer. To select the clock frequency, see "Defining Events" on page 424.

You can select from the following frequencies:

- 32 Mhz Selects the 32 MHz oscillator.
- 16 Mhz Selects the 16 MHz oscillator.
- 8 Mhz Selects the 8 MHz oscillator.
- 4 Mhz Selects the 4 MHz oscillator.
- 2 Mhz Selects the 2 MHz oscillator.
- 1 Mhz Selects the 1 MHz oscillator.
- External Selects the external clock
- Custom selects the programmable clock.
- Bus Clock selects the emulator clock, the bus clock of the emulating MCU.

If you select External, make sure to connect the TT_OSC clip (white) of the pod B cable to the external clock source.

NOTE If you specify a custom clock speed, be aware that the FSICEBASE can provide clock speeds from 4100Hz to 40MHz in steps of 5kHz. The FSICEBASE uses a clock sythesis chip to generate the clock speed. This method is not as accurate as a crystal: only within about 0.75% absolute frequency with about 5% jitter. If you choose a clock value (either from one of the radio buttons or by typing it in) that is an integer divisor of 32MHz, you will get an accurate crystal-sourced clock.

Collecting Bus Data

To instruct the Bus State Analyzer (BSA) to start collecting data:

1. From debugger main menu, select **FSCICEBASE-HC08**
2. Select **Arm Trace** — the BSA begins to collect data when the debugger starts execution of the loaded application. The BSA uses the events that you defined in the Bus State Analyzer Configuration dialog box.

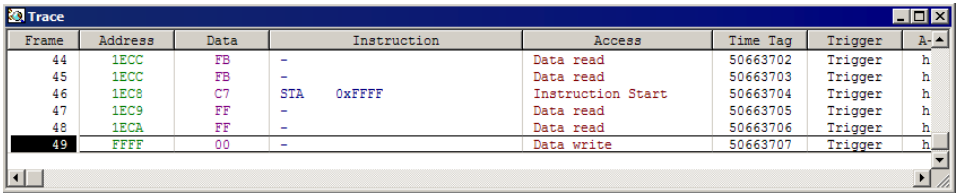
Viewing Data

You can view the data collected by the BSA in several formats. You can view: raw data, disassembled instructions, mixed raw data and disassembled instructions, and source code.

To view data:

1. From debugger main menu, select **FSCICEBASE-HC08**
2. Select **Trace** — the **Trace** window opens (Figure 17.11 on page 429)

Figure 17.11 Debugger Trace Window


Trace
Frame Address Data Instruction Access Time Tag Trigger A-
44 1ECC FB - Data read 50663702 Trigger h
45 1ECC FB - Data read 50663703 Trigger h
46 1EC8 C7 STA 0xFFFF Instruction Start 50663704 Trigger h
47 1EC9 FF - Data read 50663705 Trigger h
48 1ECA FF - Data read 50663706 Trigger h
49 FFFF 00 - Data write 50663707 Trigger h

3. To change the kinds of data and the way that data is displayed:
   a. Place mouse cursor over Trace window
   b. Right-click mouse — Menu appears allowing you to change various aspects of the Trace window

The Trace window can display trace buffer contents as raw bus cycles, as disassembled instructions, as mixed instructions and raw bus cycles, or as source code.

18

MMDS0508 Emulator

MMDS General Description

The M68MMDS05/08 Motorola Modular Development System (MMDS) is a Freescale tool for developing embedded systems based on an MC68HC08 microcontroller unit (MCU). A modular emulation system, the MMDS provides interactive control of a microcontroller application when connected to your debugger target system.

The MMDS environment allows for source-level debugging and simplifies writing and debugging code. These features significantly reduce development time.

A complete MMDS includes a station module, an emulation module (EM), and a target cable assembly. The EM completes MMDS functionality for a particular Freescale MCU or MCU family.

MMDS Features

MMDS features include:

- Real-time, non-intrusive, in-circuit emulation
- Real-time bus state analysis
- MC68HC11K1 system controller, for fast command transfer
- Compliance with ECC92 European electromagnetic compatibility standards
- Four complex data breakpoints, each qualified by an address, an address range, data, or externally connected logic clips.
- 32 variables or real-time variables, plus a 32-byte block of real-time memory, mappable anywhere within a 1-kilobyte window over the 64-kilobyte HC05/HC08 memory map.
- 64 kilobytes of emulation memory, to accommodate the largest available ROM size of current HC05/HC08 MCUs.
- 64 hardware instruction breakpoints over the 64-kilobyte memory map
- A personality file for each EM. (Each personality file defines a memory-map.)
- Latch-up resistant design (47-Ω series resistor on I/O connections to the target system) to make power-up sequencing unimportant.

- Built-in bus state analyzer that includes:
- 8K x 64 real-time trace buffer
  - Four hardware triggers, for controlling real-time bus analysis and to provide breakpoints
  - Nine triggering modes
  - Display of real-time trace data as raw data, disassembled instructions, raw data and disassembled instructions, or assembly-language source code
  - As many as 8190 pre- or post-trigger points
  - Trace buffer that can be filled as you single-stepping through user software
  - 16-bit time tag, or an optional 24-bit time tag that sacrifices eight logic clips
  - Eight software selections for the time tag clock source, permitting wide time variance between analyzer events
  - 16 general-purpose logic clips, five of which can be used to trigger the bus state analyzer sequencer
- Four software-selectable internally generated oscillator clock sources
- Built-in power supply with 85 to 264 VAC input
- RS-232 operation speeds as high as 57600 baud
- Compact size: 15.38 inches (390.6 mm) deep, 10.19 inches (258.83 mm) wide, and 2.75 inches (69.85 mm) high. The station module weighs 6.0 pounds (2.72 kg).

For connection instructions, configuration instructions, and other related information, see the installation section of this hardware addendum. For similar information with regard to EMs, see the corresponding EM user's manual.

MMDS Components

The MMDS components include:

- **Station module:** the MMDS enclosure, containing the platform board and the internal power supply. The access panel in the enclosure top lets you insert an EM easily.
- **9-lead RS-232 serial cable:** the cable that connects the station module to the host-computer RS-232 port.
- **Serial adapter:** a DB9M-to-DB25F RS-232 adapter, for use with a 25-pin host-computer serial port.

- **Two logic clip cable assemblies:** twisted-pair cables that connect the station module to your target system, a test fixture, a clock, an oscillator, or any other circuitry useful for evaluation or analysis. One end of each cable assembly has a molded connector, which fits into the pod A or pod B connector of the station module. Leads at the other end of each cable terminate in female probe tips. Ball clips come with the cables.

Separately purchased components are:

- **Emulation module (EM):** a printed circuit board that completes MMDS functionality for one or more MCUs. The two DIN connectors on the bottom of the EM fit into connectors on the top of the MMDS0508 platform board, providing power and signal connections. The EM also has a connector for the target cable. Each EM comes with its own user's manual.
- **Target cable:** a flat, flexible cable that connects the MMDS to the target system. The cable's emulator terminator mates to the EM connection; the cable's head terminator mates to a target head adapter. (A target cable and target head adapter make up a target cable assembly.)
- **Target head adapter:** a connection-cable adapter that plugs into the MCU socket of the target system. (For some EMs, the target head adapter plugs into a surface-mount adapter.)

MMDS Installation

A complete Freescale MMDS installation consists of:

- Configuring the platform board,
- Configuring the emulation module (EM),
- Installing the EM, and
- Making system cable connections.

This section explains all items except EM configuration. As EM configuration is specific to each EM, you must follow the instructions of your EM user's manual. Additionally, this sections explains how to remove an EM from the station module.

Figure 18.1 on page 434 shows the right side of the MMDS station module, with the access panel open. Callouts indicate the reset switch and power LED (on the front of the station module), and the logic cable A and B connectors (on the right side of the station module). (Alternate names for the logic cable connectors are pod A and pod B.) If you use logic clip cables, always attach the black clip to ground.

Figure 18.2 on page 434 shows the left side of the station module, with the access panel closed. Callouts indicate the power cord socket, the power switch, and the 9-pin RS-232 serial connector. The circular, +5 V out connector is reserved for possible future features. (A spacer covers the final enclosure cutout, for a future connector.)

Figure 18.1 M68MMDS0508 Station Module (Right Side)

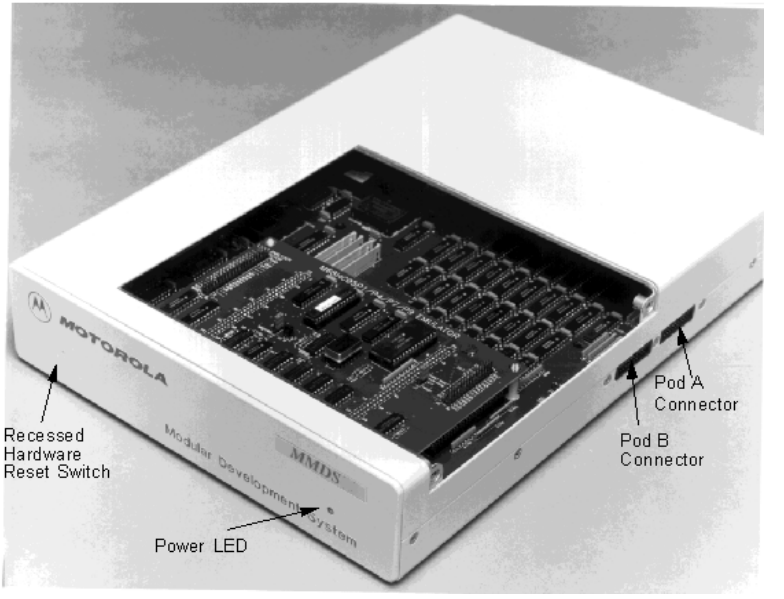
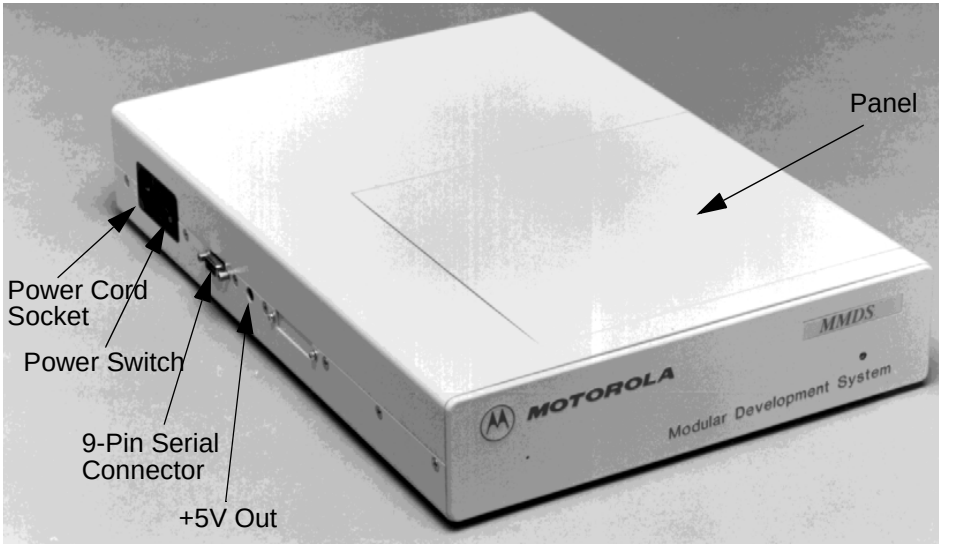


Figure 18.2 M68MMDS0508 Station Module (Left Side)



Configuring the Platform Board

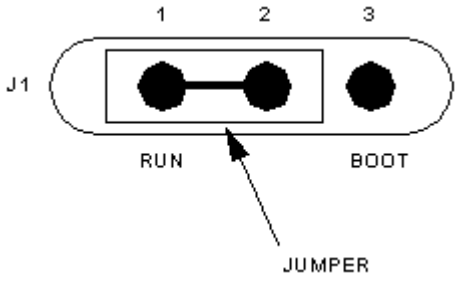
The MMDS platform board has four jumper headers, all located near the front. Jumper header J1 is for factory test. Jumper headers J2, J3, and J4 control the voltage levels for ports A through D.

NOTE Before shipping the MMDS, factory personnel configure the platform board correctly for virtually all users. You should not reconfigure platform-board headers unless your EM user's manual tells you to.

Factory Test Header (J1)

Figure 18.3 on page 435 shows the factory configuration of jumper header J1. The jumper setting between pins 1 and 2 is correct for MMDS operation.

Figure 18.3 Factory Test Header (J1)



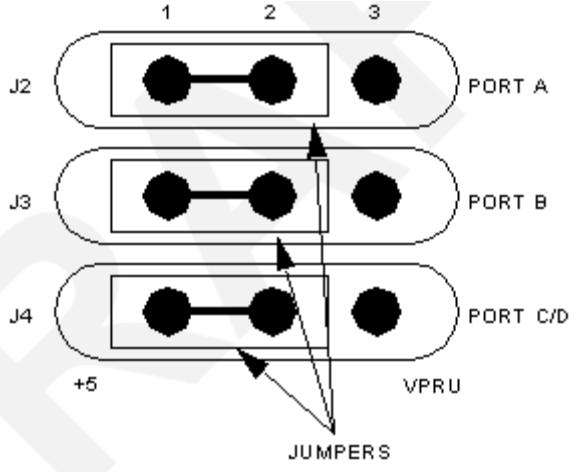
NOTE The alternate jumper position is reserved for factory tests.

Port Voltage Control Headers (J2–J4)

Jumper headers J2 through J4, near the right front corner of the platform board, set the voltage levels for ports A through D. Header J2 is the port A control; header J3 is the port B control; header J4 is the control for port C or D (whichever pertains to your EM).

Figure 18.4 on page 436 shows the factory configuration. The jumpers between pins 1 and 2 of these headers set the +5-volt level for all ports. This is the correct configuration for MMDS operation, unless your EM user's manual says that your EM is a low-voltage board.

Figure 18.4 Port Voltage Control Headers (J2–J4)



NOTE If your EM can operate at low voltage, you can configure any of the ports for the low-voltage level. To do so, reposition the corresponding header's jumper to pins 2 and 3.

Installing the EM

CAUTION Be sure to turn off power to the MMDS when you install or remove an EM. This prevents power surges that could damage MMDS circuits.

To install an EM in the MMDS enclosure:

1. Make sure that station module power is off.
2. Unscrew (one quarter turn) the two captive screws of the access panel, then remove the panel.
3. Install the EM on the platform board: Carefully fit the female 96-pin DIN connectors (on the bottom of the EM) onto the corresponding male DIN connectors (on the top of the platform board). Snap the EM onto the plastic standoffs and make sure that the DIN connectors are joined firmly.
4. Connect the target cable, if appropriate.
5. Reposition the access panel, securing it with the screws.

NOTE Many EM boards have 64-pin female DIN connectors. Like their 96-pin counterparts, these smaller connectors also mate with the male DIN connectors of the platform board. The connector keys ensure proper alignment.

Removing the EM

To remove an EM from the MMDS enclosure:

1. Make sure that station module power is off.
2. Unscrew (one quarter turn) the two captive screws of the access panel, then remove the panel.
3. Disconnect the target cable from the EM target connectors.
4. Unsnap all plastic standoffs from the edges of the EM.
5. Carefully lift the EM straight up, separating it from the platform board.
6. You are ready to install a different EM or to configure the platform board. When you finish such other actions, reposition the access panel, securing it with the screws.

Making Cable Connections

You must connect the station module to the host computer and to line power. According to your specific application, you also may need to connect the EM to your debugger connection, or to connect the logic clip cable assemblies.

Host Computer Connection

Connect the 9-lead serial cable between the MMDS 9-pin serial connector and the COM1 serial port connector of the host computer.

- COM1 is the default serial port. You may use a different host serial port, provided that you use the appropriate software startup command to specify the port number.
- A 9-to-25-pin adapter came with your MMDS. If the host serial port is a 25-pin connector, use this adapter between the port connector and the serial cable.

Bus State Analyzer Connection

If your work session includes bus state analysis, you may need the logic clip cable assemblies. The two logic clip connectors, pod A and pod B, let the analyzer capture external events. Logic clip connections also let you input external clock signals for the emulator and analyzer.

The pod A and pod B connectors are on the right side of the station module. Pod A is nearest the station module power supply. These pod connectors correspond to the cable A and cable B selections available in the bus state analyzer configuration window. The Connector Information section includes pinout information for both logic clip connectors.

If you need only one logic cable assembly, connect it to either pod A or pod B. Orient the cable connector so that its pin 1 connects to pin 1 of the pod, according to the keyed plastic. Connect the other end of the logic cable assembly to an external target point. Optionally, connect the probe tips to the ball clips that come with the cable assembly, then connect the ball clips to appropriate external test points.

NOTE Always connect the black (ground) probe tip to an appropriate connection-system ground point before connecting other clips to target points. The pod A white probe is the external clock input for the emulator; the pod B white probe of pod B is the external clock input for the analyzer.

If you need the second logic cable assembly, connect it in the same way to the remaining pod connector of the station module. Make connection-system connections as for the first cable.

Target Cable Connection

To connect the MMDS to a debugger connection (target system), you must use a target cable assembly: a target cable and a target head.

CAUTION Press only on the rigid plastic terminators of the cable. Pressing on the flexible part of the cable can damage the cable.

Follow these instructions:

1. Make sure that station module power is off; make sure that power is *not* applied to the target system.
2. Remove the MMDS access panel, for access to the EM.
3. The EM has one or two target connectors, on its right side. Connect the target cable's emulator terminal to the EM target connectors.
4. Connect the target cable's head terminator to the appropriate target head adapter.

NOTE The EM user's manual should identify the target head adapter appropriate for your target system. Make sure that the adapter and the target cable mate correctly.

5. Plug the target head adapter into the MCU socket (or surface-mount adapter) of the target system.

6. Replace the access panel. (The target cable must run through the slit in the station-module enclosure.)

Power Connection

The final MMDS connection is line power. The MMDS power switch is the rocker switch on the left side of the station module.

1. Set the power switch to OFF.
2. Insert the female end of the power cord into the power cord socket.
3. Plug the other end of the cord into a line-power outlet.
4. Set the power switch to ON.

The green LED on the front of the station module lights to confirm system power.

Connector Information

This section provides pin assignments and signal descriptions for connectors common to all MMDS systems. For similar information about EM connectors, see the corresponding EM user's manual.

Additionally, this section explains the reset switch, and fuse replacement.

Reset Switch

RS-232 handshake signals control MMDS resets. A reset initializes the control board from its startup point. If the host serial port does not implement handshaking, you must reset the MMDS manually.

To do so, find the reset switch, which is recessed behind the small hole in the front of the station module. Insert a probe or stiff wire into the reset switch hole. Press gently to trip the switch.

RS-232 Serial Connector

Figure 18.5 on page 440 shows pin numbering for the MMDS serial connector. Table 18.1 on page 440 lists the signals transmitted on the 9-lead serial cable.

Figure 18.5 RS-232 Serial Connector

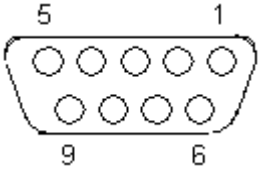


Table 18.1 RS-232 Serial Connector and Cable Pin Assignments

Connector Pin	Mnemonic	Signal
1	DCD	DATA CARRIER DETECT — Output signal that indicates detection of an acceptable carrier signal.
2	RX	RECEIVE DATA — Serial data output line.
3	TX	TRANSMIT DATA — Serial data input line.
4	DTR	DATA TERMINAL READY — Input signal that indicates on-line/in-line/active status.
5	GND	GROUND
6	DSR	DATA SET READY — Output signal that indicates on-line/in-line service/active status.
7	RTS	REQUEST TO SEND — Input signal that requests permission to transfer data.
8	CTS	CLEAR TO SEND — Output signal that indicates a ready-to-transfer data status.

Logic Cables and Connectors

The diagram below shows the pin numbering for both pod A and pod B logic cable connectors. Table 18.2 on page 440 lists the pin assignments.

Table 18.2 Pods A and B Logic Cable Pin Assignments

19									1
•	•	•	•	•	•	•	•	•	•

Table 18.2 Pods A and B Logic Cable Pin Assignments

•	•	•	•	•	•	•	•	•	•
20									2

Table 18.3 . Pod and Logic Cable Pin Assignments

Pod Pin	Pod A Signal	Pod B Signal	Probe Color
1	LC0	LC8	Brown (BRN)
2	GND	GND	
3	LC1	LC9	Red (RED)
4	GND	GND	
5	LC2	LC10	Orange (ORG)
6	GND	GND	
7	LC3	LC11	Yellow (YEL)
8	GND	GND	
9	LC4	LC12	Green (GRN)
10	GND	GND	
11	LC5	LC13	Blue (BLU)
12	GND	GND	
13	LC6	LC14	Purple (PUR)
14	GND	GND	
15	LC7	LC15	Gray (GRY)
16	GND	GND	
17	EXT_OSC	TT_OSC	White
18	GND	GND	
19	GND	GND	Black
20	GND	GND	

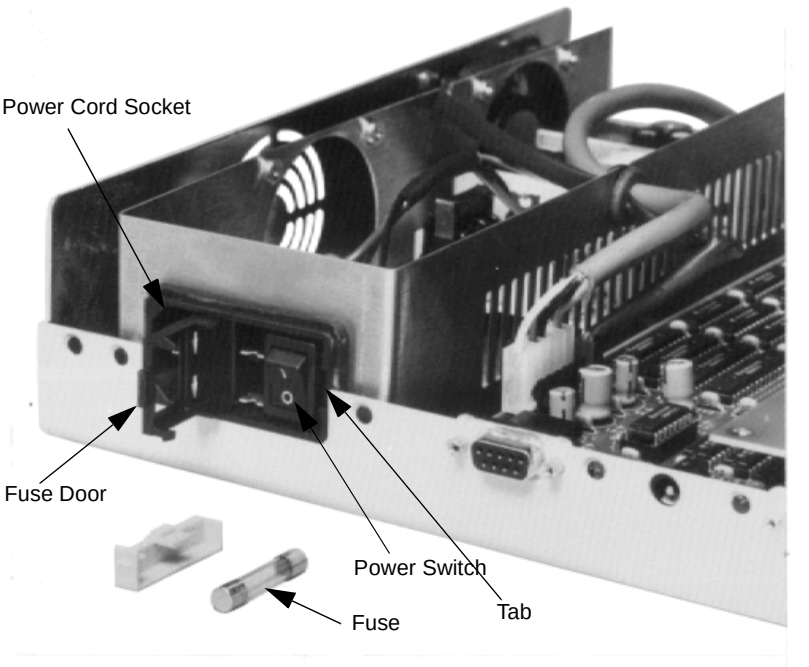
Note the folloeing points with regard to the preceding tables:

- Pins 19 of both pods provides connection to an external ground.
- Pod A pin 17 is the external clock input for the emulator. To use this source, make the desired clock connection to the white probe tip, then use the OSC command to select an external source.
- Pod B pin 17 is the external timetag input for the bus state analyzer. To use this source, make the desired clock connection to the white probe tip, then use the TIMETAG command to select an external time tag source for the analyzer.

Power Supply Fuse Replacement

The station module power switch/connector assembly contains a standard 1/4 x 1 1/4 inch, 1.6-ampere, 250-volt ceramic, time-delay fuse. Figure 18.6 on page 442 shows this assembly with its door open (for fuse replacement).

Figure 18.6 Power Switch/Connector Assembly



To replace the fuse:

1. Press the power switch OFF and disconnect the power cord.
2. Insert a small screwdriver at the tab on the right edge of the switch/connector assembly. (Figure 1 shows where to insert the screwdriver.) Gently pry open the assembly door, which swings open to the left.
3. Remove the fuse holder from the switch/connector assembly. Remove the fuse from the holder.
4. Insert the replacement fuse into the holder. Then re-install the holder in the switch/connector assembly. Make sure that the fuse holder arrow points down. Close the assembly door.
5. Reconnect the power cord. This completes fuse replacement.

MMDS Connection

An advanced feature of the debugger for the embedded systems development world is the ability to load different debugger connections (target components), which implement the interface with Debugger connections. This document introduces the MMDS0508 Freescale Modular Development System for the MC68HC05 and MC68HC08 MCU families.

The MMDS is a Freescale interface that the debugger uses to communicate with an external system (also called a *target or target system*).

This chapter describes MMDS0508 features, including memory mapping and the bus analyzer.

With this interface, you can download an executable program from the debugger environment. The destination of this program is an external target system, based on a Freescale MCU, that executes the program. The debugger receives feedback of real target-system behavior.

The debugger fully supervises and monitors the target MCU. That is, the debugger controls the CPU execution. You can read and write in internal or external memory (even when the CPU is running); you can single-step, run, or stop processes in the CPU.

NOTE **Uninvolved Components:** As an external MCU executes the code, the MMDS connection cannot provide memory statistics. This means that you cannot use the MMDS connection for profiling, coverage analyzing, watchpoints, or I/O Simulation.

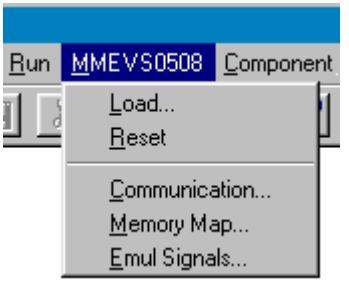
MMDS0508

The MMDS0508 is an emulator system, for CPU05 or CPU08 MCUs, that provides emulation memory and a bus state analyzer.

Configuration

The functions above are specific for the MMDS0508, so are available only with this emulator component. To use these features select the corresponding menu commands shown in Figure 18.7 on page 444, such as **MMDS0508 > ...**

Figure 18.7 MMDS0508 Menu



These menu functions are part of the MMDS0508 (Freescale) Modular Development System. Their supporting libraries consist of dialog boxes and routines to interface the debugger and the hardware.

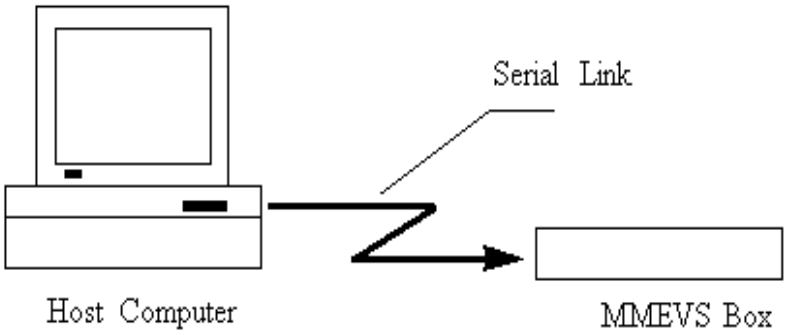
Interfacing Your System and the Connection

The MMDS0508 box connects to an RS-232 serial port of your system.

Hardware Connection

Use the cables that came with the MMDS0508 to connect the host computer to the MMDS0508 box (the diagram below depicts this connection). Configure the host computer to be a data terminal, so that it sends data on the TxD lead and receives data on the RxD lead, as shown in Figure 18.8 on page 445.

Figure 18.8 Hardware Connection

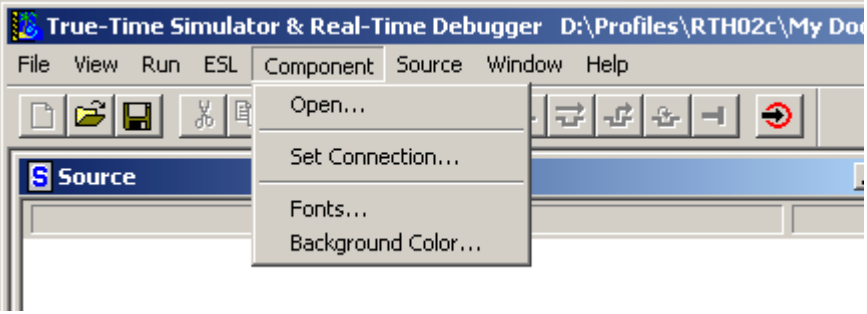


Loading the MMDS0508 Connection

Usually, the `PROJECT.INI` file specifies the connection. To make the MMDS the connection, change the file's "`Target=`" line to "`Target=Motosil`". The MotoSIL driver then automatically detects the MMDS. However, if the driver detects nothing, an error message informs you that the target is not connected to the expected port. Additionally, the *Communications Device Specification* dialog box appears, so that you can set the correct baud-rate and communication-port parameter values. See the *Communication Configuration* section of this document for more details.

Another way to load the MMDS0508 connection is selecting **Set Connection...** from the *Component* Menu (below) as shown in Figure 18.9 on page 445, then choosing **MotoSIL** from the list of possible connections.

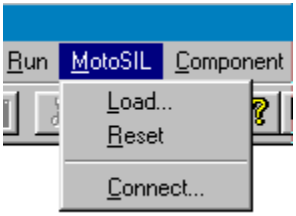
Figure 18.9 Set Connection Option of the Component Menu



The MotoSIL driver automatically tries to find the MMDS connection, behaving as explained above.

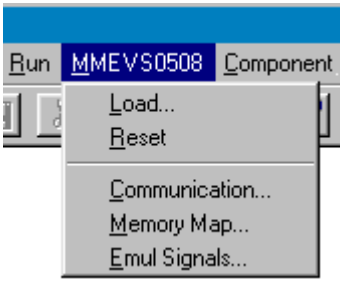
If MotoSIL does not detect a connection, the *MotoSIL* item remains in the main menu bar as shown in Figure 18.10 on page 446:

Figure 18.10 Loading MotoSIL Connection



But after successful connection loading, the *MMDS0508* menu replaces the *Connection* or *MotoSIL* menu in the main menu bar as show in Figure 18.11 on page 446.

Figure 18.11 Loading MMDS Connection

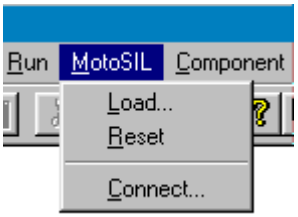


Communication Configuration

In most situations, the debugger uses its default values to set communication with the MMDS automatically. In case of any problems, the dialog box show in Figure 18.12 on page 447 appears, so that you can correct settings.

NOTE Another way to open this dialog box is by selecting **MotoSIL > Connect...** from the menu bar. This method is appropriate if previous connection attempts failed and MotoSIL still is in the main menu.

Figure 18.12 MotoSIL Communication Configuration

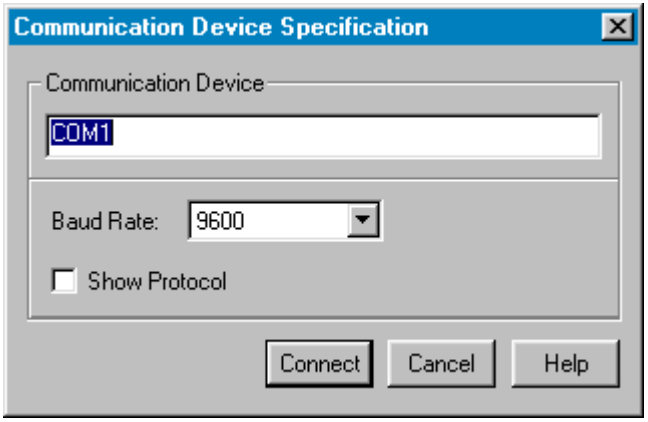


Make sure that host-computer parameter values are correct; make sure that the serial-communication setting is correct. Otherwise, communication between the debugger and the connection is not possible.

Communication Device Specification

If the host and connection are not connected, or if the communication device is not properly indicated, the *Communications Device Specification* dialog box appears as shown in Figure 18.13 on page 447 :

Figure 18.13 Communication Device Specification Dialog Box



Type the name of an available communication device in the *Communication Device* edit box, use the drop-down control to set the baud rate, then click *Connect*. (The default communication device is COM1.)

Once connection succeeds, the debugger saves the settings as defaults for later debug sessions. Should the connection fail, a message box so informs you, so that you can define a new communication device. To quit the dialog box and the environment, click *Cancel*.

NOTE Saving the communication device and the baud rate using this dialog box overrides environment variables BAUDRATE and COMDEV of the default.env file.

Data Format

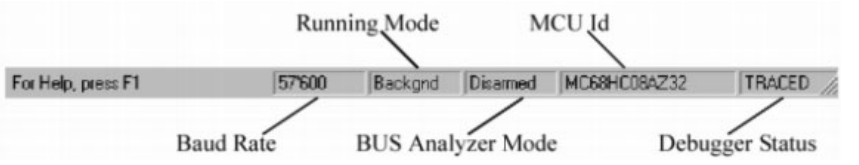
The MMDS0508 data format is 8 data bits, 1 stop bit, no parity, and a variable baud rate. The default speed is 9600 baud, unless you change this default via the menu selection **MMDS0508>Communication...**.

Communication speeds of 1200 through 115200 baud are available, depending on the host-computer hardware.

MMDS Debugger Status Bar

Once you have loaded the MMDS Connection, the debugger status bar gives specific information as shown in Figure 18.14 on page 448:

Figure 18.14 Debugger Status Bar

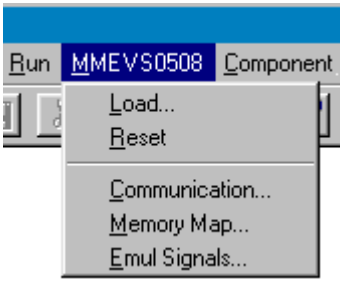


From left to right, this information is: the serial-communication baud rate, the debugger running mode, the BUS analyzer mode, the MCU name (depending on the MCU-Id), and the debugger status.

MMDS0508 Menu Options

The Figure 18.15 on page 449 shows the MMDS0508 Menu Entries:

Figure 18.15 MMDS Menu



Communication Baud Rate

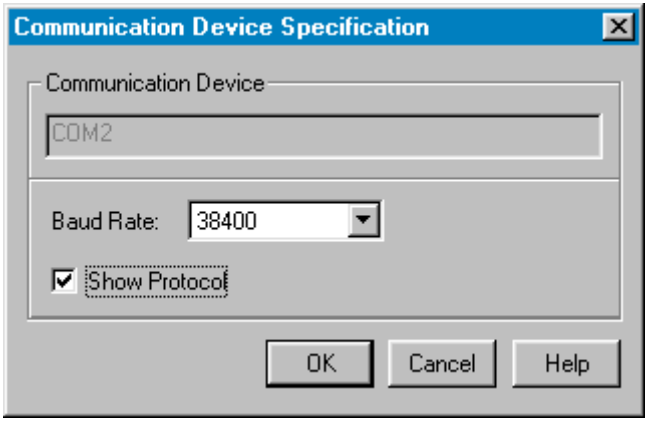
You should specify the baud rate for host-computer-to-MMDS0508 communication early in a session. The system operates most efficiently at the maximum baud rate that the host computer supports.

You can modify this baud rate, as text below explains.

Communication Option

Select **MMDS0508>Communication...** to display the *Communication Device Specification* dialog box as shown in Figure 18.16 on page 450. Use the drop-down control to specify the maximum value (115200 baud), or to specify the maximum rate your host supports. If communication fails, the debugger reverts to the previous successful baud rate.

Figure 18.16 Communication Device Specification Dialog Box



Maximum Baud Rate

The maximum baud rate depends on the speed and interrupt load of the host computer. For slow book computers, or for computers running in a network, the maximum baud rate may be as low as 19200. A buffered I/O card may allow the maximum rate of 115200 for any host computer. The default value is 9600.

Show Protocol

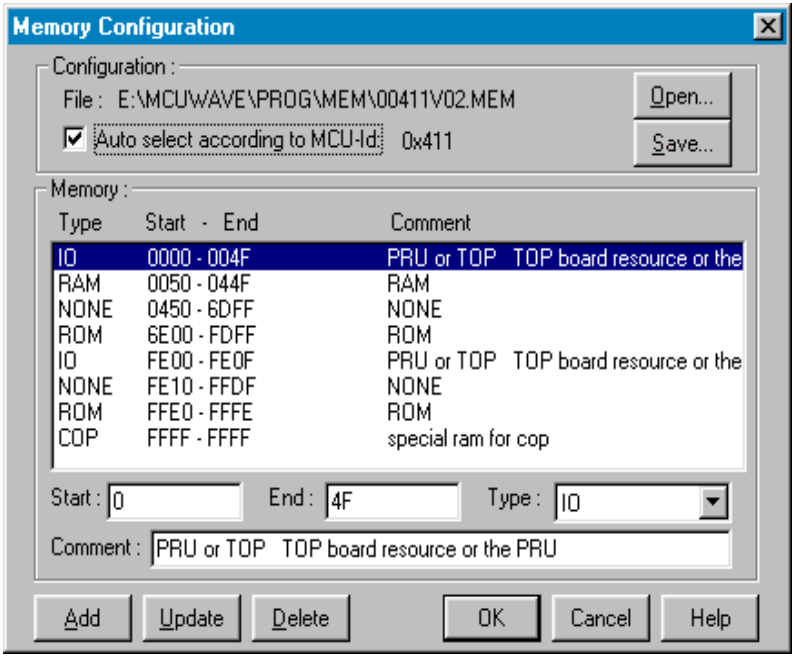
If you check the *Show Protocol* checkbox, the system reports all commands and responses in the command line window.

NOTE Freescale support personnel use this feature.

Memory Map Option

To view the memory layout, choose **MMDS0508 > Memory Map...** This opens the *Memory Configuration* dialog box shown in Figure 18.17 on page 451.

Figure 18.17 Memory Configuration Dialog Box



This dialog box shows the connection's memory setup. The system automatically loads this setup if you check the *Auto select according to MCU-Id* checkbox. The debugger identifies and sets the memory map through the processor MCU-Id. To open another configuration, click the *Open* button. To save modifications to the current configuration, click the **Save...** button.

Personality (.MEM) Files

For proper operation, the Motosil connection must load the personality file (`.MEM` file) that matches the connected Emulation Module (EM).

The `.MEM` file filename format is:

```
0nnnnVxx.MEM
```

where '`nnnn`' is the four-digit, hexadecimal 'MCU-Id' number of the MCU, and '`xx`' is a two-digit version number.

If the connection cannot find this file, or if the file is not valid, the *Error* message box appears as shown in Figure 18.18 on page 452:

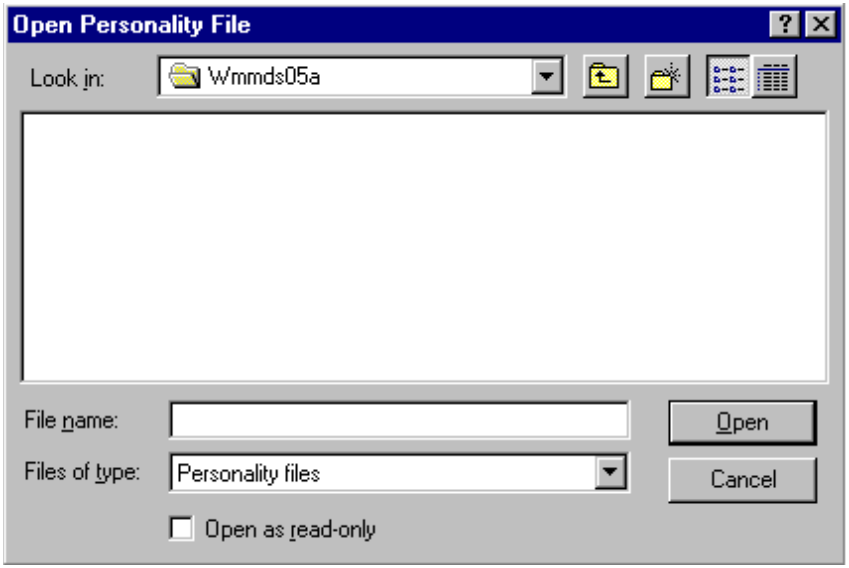
Figure 18.18 Error Message Box

Error
Personality file for Id 'A18' was not found in directory
'C: \MCU08\PROG\MEM'. If a matching .MEM file is available click 'Retry' to
select and install this file in the directory 'C:\MCU08\PROG\MEM'. If such a
file is not available, proper operation will not be possible.
Retry
Cancel

Click *Cancel* to open the *Communication Device Specification* dialog box, instead of establishing the connection.

Click *Retry* to bring up the *Open Personality File* dialog box shown in Figure 18.19 on page 452:

Figure 18.19 Open Personality File Dialog Box



The OpenPersonality File dialog box lets you browse to find and open the necessary `.MEM` file.

If you select another invalid `.MEM` file, the error message box and the *Configuration Device Specification* dialog box reappear.

If the `.MEM` file is valid, the connection loads the file, copying it into the `\PROG\MEM` directory, with ‘`V00`’ as the version number (for example, `00A18V00.MEM`).

NOTE The *Memory Configuration* dialog box displays the current memory map.

When starting the debugger:

- If you have checked the *Auto Select according MCU-ID* checkbox, the system automatically loads the default personality file for the MCU-ID. If this checkbox is clear, the system automatically loads the most recently opened or saved memory map file.
- You can use the *Memory Configuration* dialog box to modify the memory configuration, then save this new configuration into a memory configuration file. (Click the *Save* button.)
- You can use the *Memory Configuration* dialog box to load a different memory configuration file (one that you previously defined and saved). (Click the *Open* button.)
- Another way to load personality files (or any memory configuration files) is to enter the `LOADMAP` command in the command line.

Dual-Port RAM

The *Dual-Port RAM* area lets you specify the base address and enable “Real-Time Memory”.

To specify the base address, enter the desired value in the *Base Address* edit box. To Enable the “Real-Time Memory”, check the *Enable* checkbox. The size of the “Real-Time Memory” is frozen at 1 kilobyte.

Also see the *M68MMDS05/08, Motorola Modular Development System, User's Manual* for further information.

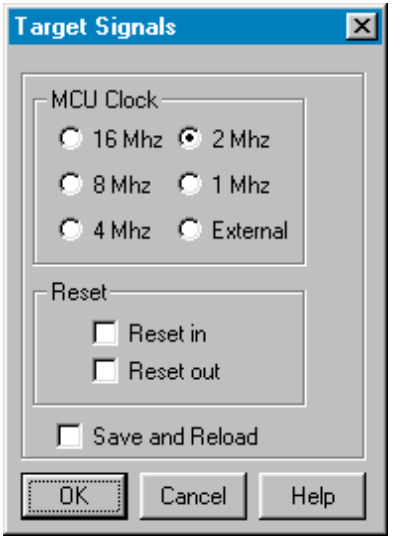
Memory

The *Memory* area lets you specify the “Real-Time Memory”. For the MMDS0508, this is dual-ported memory that you can assigned to any valid RAM or ROM address. While the MMDS0508 is running, the debugger can display and modify this “Real-Time Memory.” However, if a portion of this memory overlays internal MCU I/O, RAM, or EEPROM, the *Memory Configuration* dialog box can only display, not monitor, that memory portion.

Signals Emulation Option

To specify MMDS0508 emulator signals, choose the **MMDS0508>Emul Signals...** menu selection. This opens the *Target Signals* dialog box show in Figure 18.19 on page 452.

Figure 18.20 Target Signals Dialog Box



This dialog box lets you specify the MCU clock and the reset signal connection. Warning: in order to use any of these clock signals, you must configure EM jumper headers correctly.

This dialog box displays settings that the system reads from the MMDS0508. Click *Ok* to close the dialog box, and to write values back to the MMDS0508. Check the *Save and Reload* checkbox to have the system save the configuration, then reload this configuration the next time you start the debugger.

MCU Clock

The *MCU Clock* area lets you specify a different MCU clock, provided that the EM configuration is correct.

Reset

The *Reset* area lets you specify the reset-signal connection with the target system.

Bus Tracing

To run the Bus Analyser, choose the **MMDS0508>Bus Trace** menu selection. Please see the *Bus Analyser* section of this document for details.

Default Connection Setup

As with any debugger connection, you can use the *Component* menu to load the MMDS connection, or you can set the MMDS connection as a default in the `PROJECT.INI` file. This file should be in the project directory.

Example of `PROJECT.INI` File:

```
[DEFAULTS]
Window0=Source      0    0   50   40
Window1=Assembly   50    0   50   40
Window2=Register   50   40   50   30
Window3=Memory     50   70   50   30
Window4=Data        0   40   50   25
Window5=Command     0   65   50   20
Window6=Module      0   85   50   15
Target=Motosil

[Motorola ESL]
COMDEV=COM2
BAUDRATE=57600
SHOWPROT=1
```

NOTE For more information about the `PROJECT.INI` file, see the debugger manual.

Freescale ESL Parameters

In normal use, you set these parameters in the `project.ini` file once, interactively, during installation. You use these parameter values in subsequent debugging sessions.

COMDEV

This parameter specifies the host-computer communication port. COM1 is the default communication device for PCs. The default for UNIX systems is `/dev/ttya`.

For a PC: Any valid communication device (`COM1`, `COM2`, etc.).

Example: `COMDEV=COM2`

For SUN: Any valid communication device (`/dev/ttya`, etc.).

Example: `comdev=/dev/ttyb`

BAUDRATE

This parameter specifies the communication baud rate between the host computer and the connection. The debugger default is 9600 baud, but you may set any of these baud rates:

`1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200.`

Example: `BAUDRAUTE=19200`

SHOWPROT

This parameter specifies whether to report commands and responses in the command line window. To have the system report the commands and responses, give this parameter the value 1. To have the system *not* do this reporting, give this parameter the value 0. (Another way to specify reporting is checking the *Show Protocol* checkbox of the *Communication Device Specification* dialog box.)

Please see the section *Communication Configuration, Communication Device Specification.*

Bus Analyzer

Except for emulating the debugger target MCU, the most important feature that a microcontroller development tool can offer is the ability to analyze program execution activities on the target MCU bus. This analysis lets you determine what is occurring in a system, without actually affecting the system.

NOTE The MMDS0508 bus analyzer shows the logical state of the MCU bus. It does not show signal hold or setup times.

Watchpoints

The MMDS0508 automatically maps available watchpoints to a proper bus analyzer trigger pattern. You may use the bus analyzer to implement watchpoints. If you do, however, you should not use the bus analyzer for other purposes!

Trace Modes

To gather pertinent bus data, you can operate the bus analyzer in different modes. The various trace modes let you choose appropriate actions to take when a certain pattern (event), or sequence of patterns, appears on the bus. To trigger the Bus analyzer, define specific bus states as terms, and select a sequence of terms as a trigger event.

Trace Buffer

The trace buffer consists of 8,192 entries (or frames), each of which stores 96 bits. When the bus analyzer is active and the emulator is running, the system strobes a frame of the selected type into the trace buffer for each bus cycle. When trigger events occur in a specified sequence, the system stores only the specified number of additional frames.

Textual or Graphic Display

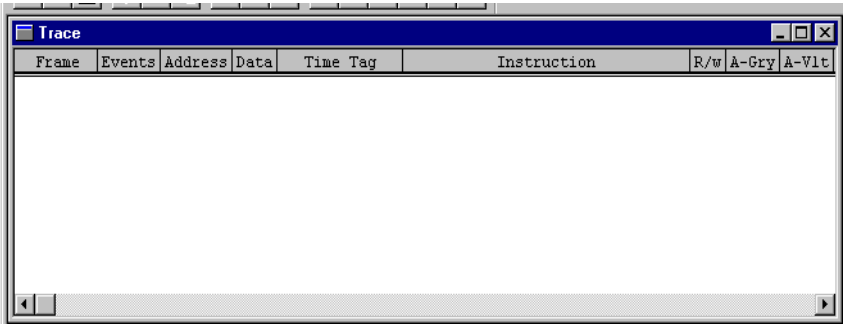
The bus analyzer displays data either textually or graphically. Use the horizontal and vertical scroll bars, as in other Windows applications, to move around the displays.

Using the Bus Analyzer

The bus analyzer functions like any debugger component, and has its own menu to control the features of the MMDS0508 Bus analyzer hardware. Another name for the Bus Analyzer window is the Trace window. This is because the bus analyzer corresponds to the trace component; you can load the bus analyzer by choosing **Open...** *Trace* from the *Component* menu. Choose the *MMDS0508* menu, then select *Bus Trace* to open the Trace Window (shown in Figure 18.22 on page 458) or the Bus Analyzer Window (shown in Figure 18.21 on page 457).

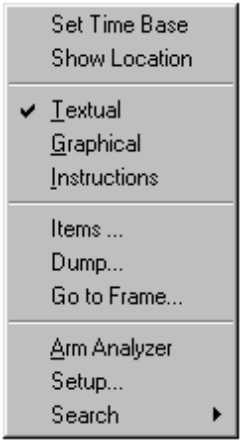
The three steps to using the Bus analyzer are: defining the data collection parameters, collecting the specified bus data (running the program), and viewing the collected data.

Figure 18.21 Bus Analyzer Window - Trace Component



Trace Group Popup Menu

Figure 18.22 Trace Group Popup Menu



The bus analyzer window has setup tabs for these functions:

- Triggers
- Search pattern
- Sequencer
- Clock timing

These tabbed setup pages are part of the *Bus Analyzer Configuration* dialog box.

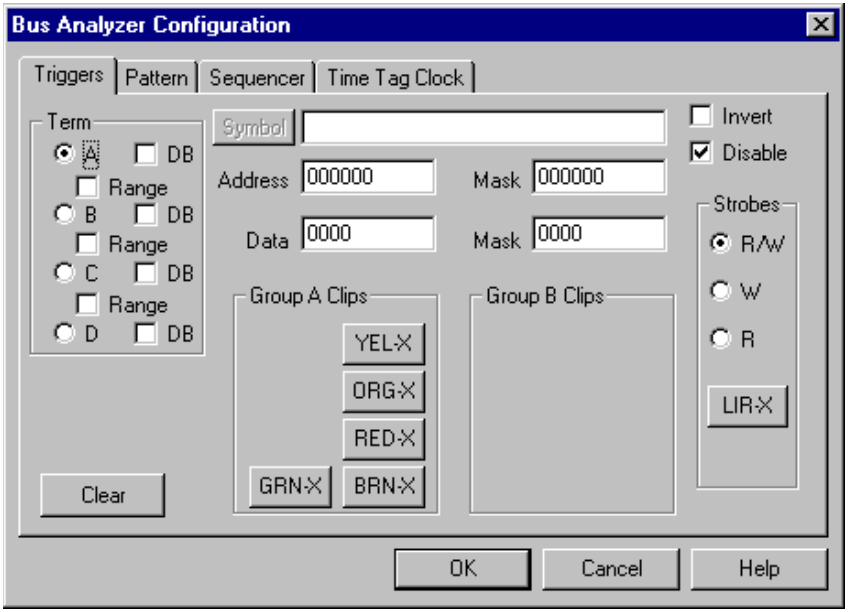
The *Bus Analyzer Configuration* dialog box lets you define symbolic names for address values. When you change setup pages in the dialog box, the address and symbolic name values must match. If there is an inconsistency, the system prompts you to:

- Use the address and remove the symbol.
- Replace address with the symbol address.
- Fix the inconsistency by returning to the dialog box and clicking the symbol button.

Trigger Setup

The sequencer mode requires one or more terms to define the trigger events. To define these events (triggers) select **Trace>Setup....** This opens the *Bus Analyzer Configuration* window as shown in Figure 18.23 on page 459 .

Figure 18.23 Bus Analyzer Configuration Window - Triggers Tab



Use the *Triggers* tab to define a trigger in terms of one, two, three, or four events. For each term, define the read-write actions and clips.

These terms can be ranged or non-ranged triggers. For ranges, you can use A and B together, and C and D together (two different ranges), or you can use B and C together (one range).

You can edit the address masks and data masks for a “don’t care” qualifier on any address or data signal. For example, if you set a trigger for address 0x1000, but the corresponding mask is 0xFFFE, then either address 0x1000 or 0x1001 will trip the trigger.

Term

Use this area to specify the term displayed for editing. The term identifies an event; that is, an event is a frame that satisfies the term.

Address

In this edit box, specify the address to which the system writes data or from which the system reads data.

Data

In this edit box, specify the data that the system is to read or write.

Strobes

Use this area to specify the read/write state on which to trigger.

Group A & B Clips

Use this area's toggle controls to specify each logic clip as High, Low, or Don't Care.

The clips buttons show the Group A logic clips with their respective colors.

You use logic clips to trace the signals of your connection as it runs under software control. When a trigger occurs, a breakpoint shows you the states of significant logic signals before, at, and after the trigger.

Invert

To specify term triggering *outside*, not inside, a defined range, check the *Invert* checkbox.

Disable

To disable the trigger for a specific term, check the *Disable* checkbox.

Clear

To clear all the changes in the *Bus Analyzer Configuration* dialog box, click the *Clear* button.

To save your trigger settings and close the dialog box, click the *OK* button.

Sequencer Setup

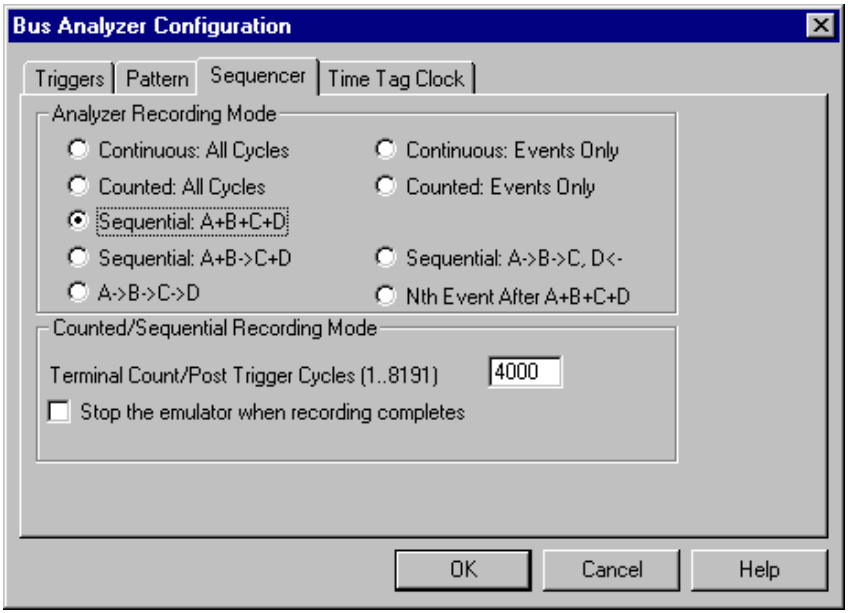
To choose the recording mode, select the menu item **Trace>Setup...** to open the *Bus Analyzer Configuration* window as shown in Figure 18.24 on page 461. Then click on the *Sequencer* tab (see the figure below).

In non-triggered (continuous and counted) modes, data collection continues until you stop the analyzer.

In triggered modes, the analyzer uses your term definitions to track the occurrence of events, then stop data collection according to some combination of the events. Each term has an associated Pre Event count that counts events for that term. The sequencer condition includes that term, once the count reaches the pre-event count.

An event is a pattern of bus signals (which can include address and data values) that logic clips and miscellaneous MCU signals connect to the analyzer. Alternatively, an event can be the negation of a defined pattern. You can define each signal to be asserted, negated, or ignored (don't care).

Figure 18.24 Bus Analyzer Configuration WIndow - Sequencer Tab



Click on one of the nine option buttons to select the recording mode.

Non-Triggered Modes

In non-triggered modes, data collection continues until you stop the analyzer.

Continuous: All Cycles

Provides a real-time, non-invasive trace of MCU bus activities. The bus analyzer stores all cycles, continuously recording bus data in the trace buffer during any emulation of the user connection. This mode does not let you define qualifications for triggering or halting data collection.

Continuous: Events Only

Stores all the events that you defined in the Triggers page.

Counted Modes

In counted modes, data collection stops when the analyzer collects the specified number of frames.

Counted: All Cycles

Records a specified number of cycles; you can trace that many cycles, of all types.

Counted: Events Only

Stores all events until it reaches the specified count. Then data collection stops.

Triggered Modes/Sequential Event Mode

In triggered modes, the analyzer uses term definitions to track event occurrence, then stop data collection according to some combination of the events. Each term has an associated Pre Event count that counts events for that term. The sequencer condition includes that term, once the count reaches the pre-event count.

The sequencer trigger includes as many as four events, in one of four sequences. Each event has a specified count; each trigger requires that specified count for each event. Each count is a minimum: the event can occur additional times. An event is a pattern of bus signals (which can include address and data values) that logic clips and miscellaneous MCU signals connect to the analyzer. Alternatively, an event can be the negation of a defined pattern. You can define each signal to be asserted, negated, or ignored (don't care).

The sequences are:

Sequential: A + B + C + D

Select this option to start bus-analyser recording after any event occurs: A, B, C, or D. Data storage ends after the specified number of post-trigger cycles.

Sequential: A -> B -> C, D<-

Select this option to start bus-analyser recording as soon as events A, B, and C occur, *in that order*. But if event D occurs before event C, the analyzer must start the whole sequence again with event A. Data storage ends after the specified number of post-trigger cycles. This mode is a simple, three-event sequence, if you do not define an event D.

Sequential: A + B -> C + D

Select this option to start bus-analyser recording when either of two sequences occurs: (1) event A, followed by either event C or event D, or (2) event B, followed by either event C or event D. Data storage ends after the specified number of post-trigger cycles. You can simplify this sequence by leaving an event undefined.

Sequential: A -> B -> C-> D

Select this option to start bus-analyser recording as soon as events A, B, C, and D occur, *in that order*. Data storage ends after the specified number of post-trigger cycles.

Nth Event After A+B+C+D

Select this option to start bus-analyser storage of data that matches events A, B, C, and D, until the analyzer stores the Nth event. Then, the analyzer stores as well the next 4096 cycles. The maximum N value is 4096, so this mode lets you store as many as 4096 events, followed by 4096 cycles.

Counted/Sequential Recording Mode

This paragraph explains how to switch beetween Counted and Sequential Recording Mode.

Terminal Count/Post Trigger Cycles (1..8191)

For a counted mode, this edit box specifies the number of bus cycles that the analyzer traces. For a sequential mode, this edit box specifies the number of post-trigger cycles that the analyzer traces. The value range for this edit box is 1 through 8191. (For a continuous modes, the analyzer ignores this value.)

Stop the emulator when recording completes

Check this checkbox to stop emulation as soon as the analyzer stops tracing bus cycles. If this checkbox is clear, emulation continues, even after analyzer data storage ends.

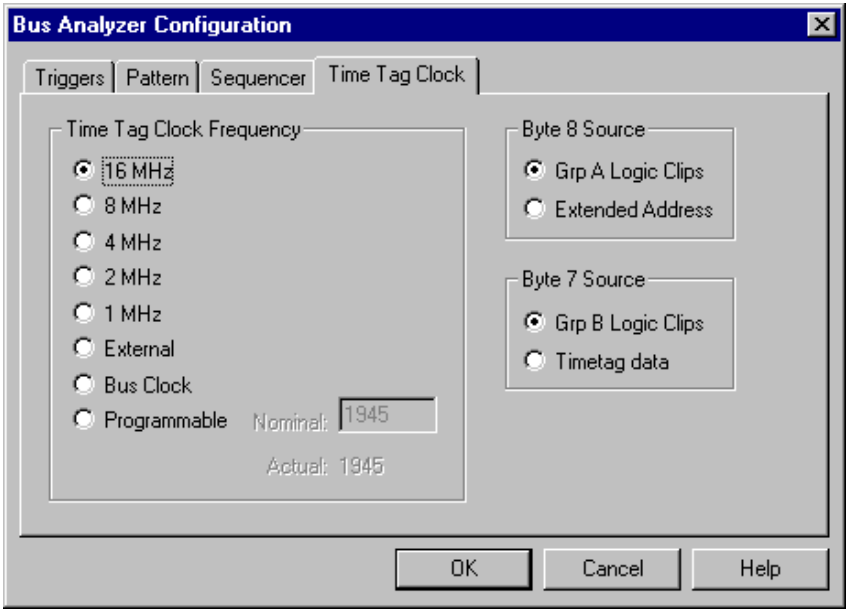
NOTE The terminal count/post trigger cycles value applies only to counted or sequential modes. For a counted mode, it is the number of cycles the analyzer stores. For a sequential mode, it is the number of cycles the analyzer stores after the trigger sequence occurs.

To save your sequencer settings and close the dialog box, click the **OK** button.

Time Tag Clock Setup

Each frame of the trace buffer includes a time reference value, or *time tag*. You can specify the clock signal that the analyzer uses for these time tags. To do so, choose the **Trace>Setup...** menu selection, to open the *Bus Analyzer Configuration* window as shown in Figure 18.25 on page 464. Then click on the *Time Tag Clock* tab.

Figure 18.25 Bus Analyzer Configuration Window - Time Tag Clock Tab



Clock Frequency

In the Time Tag Clock Frequency area, select the clock-frequency source: an internal oscillator at 16, 8, 4, 2, or 1 Mhz; an external clock; a bus clock; or a programmable clock. In general, the faster clock rates provide higher resolution, so are appropriate for faster emulator clock rates.

If you specify a programmable clock, you must enter the Hz frequency, 50 to 50,000, in the edit box. If you specify a *Nominal* value, the analyzer uses the closest, *Actual* frequency value. (A nominal value is a non-integer quotient of dividing 500,000 Hz by an integer. An actual value is an integer quotient of dividing 500,000 Hz by an integer. For example, suppose that you specify 986 in the edit box. 500,000 divided by the integer 507 yields the non-integer quotient 986.193. If you specify a nominal value, the analyzer rounds this value up to 1000, which is the integer quotient of 500,000 divided by 500.)

To save your time tag clock settings, and close the dialog box, click the *OK* button.

Collecting Data

When you have initialized the emulator and the bus analyzer has been initialized, and when you have connected any logic clips that you defined, emulation can begin.

Arming the Analyzer

Before you can start data collection, you must arm the bus analyzer. To do so, choose the **MMDS0508>Bus Trace>Arm Analyzer** menu selection. The status bar shows *Armed* status for the bus state analyzer, and the menu selection changes to *Disarm Analyzer*.

Disarming the Analyzer

To stop the analyzer, choose the **MMDS0508>Bus Trace>Disarm Analyzer** menu selection. The status bar shows *Disarmed* status for the bus state analyzer, and the menu selection changes to *Arm Analyzer*.

Start Emulation

To begin emulation, choose the debugger **Run>Continue** menu selection, or enter any other command that starts program execution. Emulation continues until a breakpoint, a watchpoint, the bus analyzer, or you stop the emulation. When emulation stops, the system updates data in the Bus Analyzer window (see below).

Status Bar

The debugger status bar shows the status of the bus analyzer. Before you arm the analyzer, the status is *Disarmed*: not ready to collect data. After you arm the analyzer, the status is *Armed*: ready to collect data. When emulation begins, the status is *Running*: the analyzer is monitoring events. When the analyzer collects data, the status is *Analyzing*.

Halt Data Collection

To halt data collection manually, choose the **MMDS0508>Bus Trace>Disarm Analyzer** menu selection. This stops data collection, but does not stop emulation.

Halt Emulation

Stopping emulation also stops data collection, but leaves the analyzer armed. Data collection resumes when emulation starts again.

Recording Bus Data

When data collection begins, the bus analyzer starts recording bus data into the buffer. When it reaches the end of the buffer, the Bus analyzer wraps around to the first buffer frame and continues recording. This process continues until you disarm the analyzer, until the analyzer records the specified number of frames, or until the analyzer records the specified number of post-trigger cycles following the trigger event.

Trigger Event

When the analyzer detects a trigger event, it latches the event cycle into the buffer and continues recording data until it collects the specified number of post-trigger cycles. Then the bus analyzer stops collecting data, and its status changes to *Disarmed*.

At the same time the analyzer records the first post-trigger cycle, it automatically begins searching for the next trigger event.

Subsequent events can occur while the bus analyzer collects post-trigger cycles for the first event. In such a case, the analyzer continues collecting post-trigger cycles, but marks the cycles for the subsequent events.

Viewing Collected Data

View Cycles

The bus analyzer provides several views of the collected cycles. At the end of cycle collection, the trace buffer contains as many as 8192 of the most recently stored frames. Those that have the highest numbers usually are the post-trigger frames. Those that have the lowest numbers are the frames stored before the trigger occurred, if any.

When the bus analyzer is not in the *Analyzing* state, it displays data in the *Trace* window.

Textual, Graphical or Instructions

To specify the type of bus-data display, select *Textual*, *Graphical*, or *Instructions* from the *Trace* popup menu.

NOTE If the sequencer did not specify collection of all frames (for example, if it specified an events-only mode) the menu may not include the Instructions selection.

The analyzer can display its contents as text or as a graphic. You can have a textual display of all frames, or a textual display of only instruction-start frames. Additionally, you can choose the **Trace>Items...** menu selection to specify items for display.

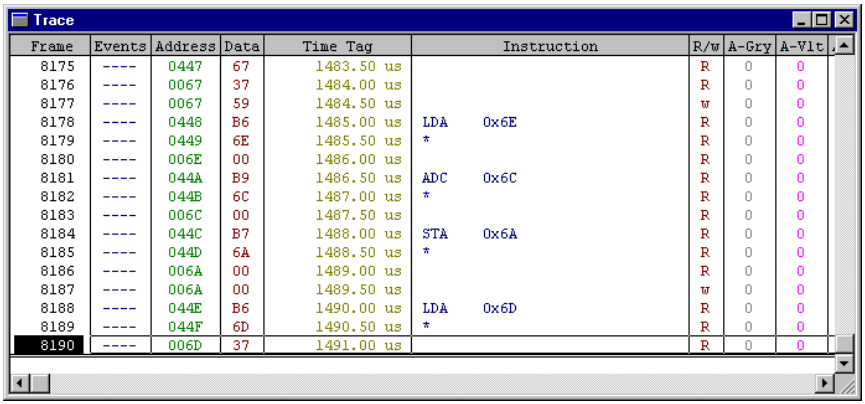
Textual Display

If you choose the *Textual* format, the software displays all the frames of the trace buffer contents, in a textual form, as shown in Figure 18.26 on page 467. Use the vertical scroll bar to scroll to other frames. Use the horizontal scroll bar to scroll to other signals.

Each display line gives the data of one frame: text below explains these data items. You can add or remove data items. (Please see the section *Adding / Removing Items in the*

Trace Window.) The marker consists of two horizontal lines, which facilitate reading the data items for a specific frame; the analyzer highlights the number of the marked frame.

Figure 18.26 Trace Window - Textual Format Display



Frame	Events	Address	Data	Time Tag	Instruction	R/w	A-Gry	A-Vlt
8175	----	0447	67	1483.50 us		R	0	0
8176	----	0067	37	1484.00 us		R	0	0
8177	----	0067	59	1484.50 us		w	0	0
8178	----	0448	B6	1485.00 us	LDA 0x6E	R	0	0
8179	----	0449	6E	1485.50 us	*	R	0	0
8180	----	006E	00	1486.00 us		R	0	0
8181	----	044A	B9	1486.50 us	ADC 0x6C	R	0	0
8182	----	044B	6C	1487.00 us	*	R	0	0
8183	----	006C	00	1487.50 us		R	0	0
8184	----	044C	B7	1488.00 us	STA 0x6A	R	0	0
8185	----	044D	6A	1488.50 us	*	R	0	0
8186	----	006A	00	1489.00 us		R	0	0
8187	----	006A	00	1489.50 us		w	0	0
8188	----	044E	B6	1490.00 us	LDA 0x6D	R	0	0
8189	----	044F	6D	1490.50 us	*	R	0	0
8190	----	006D	37	1491.00 us		R	0	0

Frame This column lists the cycle (or frame) number of each frame: identifying integers, from 1 to 8191. The most recently stored frame has number 8191 (or the highest number all the frames stored).

Events

This column lists identifiers of matching events. That is, if frame data matches any event data definitions, the analyzer displays the event identifiers (A, B, C, or D) in this column.

Address

This column lists frame address bus values, as four hexadecimal digits. Each value is the address on the address bus at the time the analyzer strobed the frame into the trace buffer.

Data

This column lists frame data bus values, as two hexadecimal digits. Each value is the values on the data bus at the time the analyzer strobed the frame into the trace buffer.

Time Tag

This column lists representations of the time tag count, stored when the analyzer strobed each frame into the trace buffer. If the time tag clock was the bus clock, this column shows the number of time-tag-clock cycles. If the time tag clock was a different clock, this column shows a number of seconds (or fractions of seconds).

Time Measuring

To assign the time-tag value 0 to a frame, position your mouse cursor over the frame entry and click the right mouse button. The *Trace* menu appears; select *Set Time Base*. The analyzer re-displays all time-tag values relative to the bus cycle of the frame you specified.

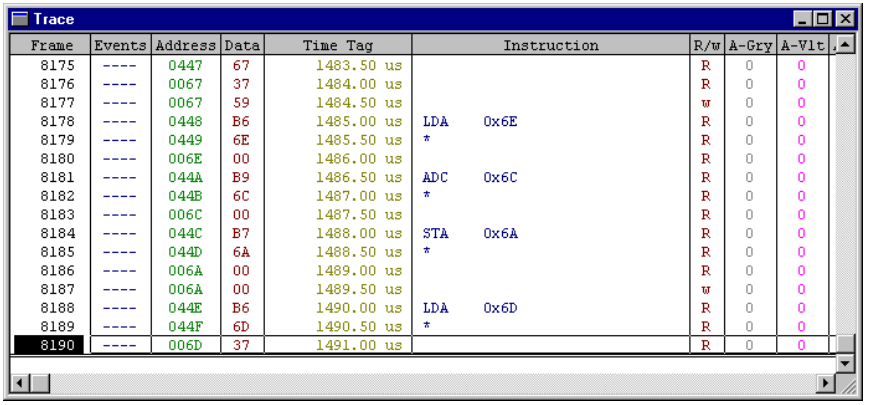
Control Signals

The remaining columns of the *Trace* dialog box show the values of control signals or of logic clips.

Instructions Only Display

If you choose the *Instructions* format, the software displays only the frames where instructions start as show in Figure 18.27 on page 468. (The *Instructions* format is not available for an *Events Only* recording mode.)

Figure 18.27 Trace Window - Instructions Only Format Display



Frame	Events	Address	Data	Time Tag	Instruction		R/w	A-Gry	A-Vlt
8175	----	0447	67	1483.50 us			R	0	0
8176	----	0067	37	1484.00 us			R	0	0
8177	----	0067	59	1484.50 us			w	0	0
8178	----	0448	B6	1485.00 us	LDA	0x6E	R	0	0
8179	----	0449	6E	1485.50 us	*		R	0	0
8180	----	006E	00	1486.00 us			R	0	0
8181	----	044A	B9	1486.50 us	ADC	0x6C	R	0	0
8182	----	044B	6C	1487.00 us	*		R	0	0
8183	----	006C	00	1487.50 us			R	0	0
8184	----	044C	B7	1488.00 us	STA	0x6A	R	0	0
8185	----	044D	6A	1488.50 us	*		R	0	0
8186	----	006A	00	1489.00 us			R	0	0
8187	----	006A	00	1489.50 us			w	0	0
8188	----	044E	B6	1490.00 us	LDA	0x6D	R	0	0
8189	----	044F	6D	1490.50 us	*		R	0	0
8190	----	006D	37	1491.00 us			R	0	0

Graphical Display

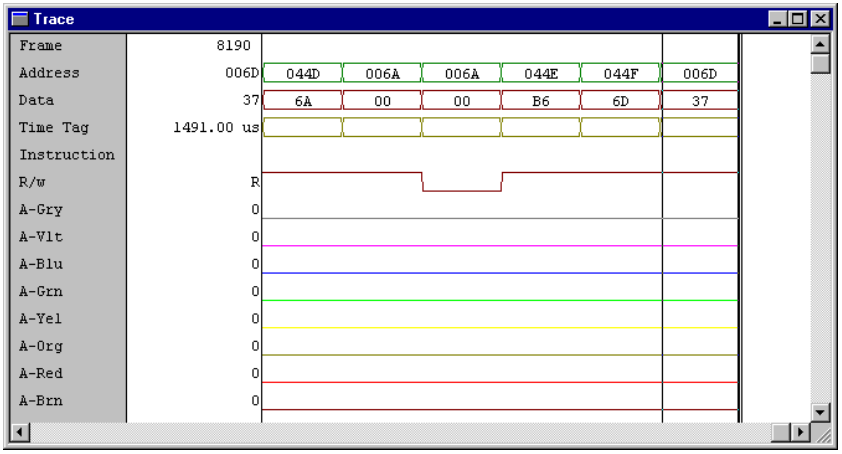
The figure below shows the graphical display of bus analyzer data. Selections of the *Trace* menu let you switch between formats easily, at any time. The left column lists data items. The next column lists the data values for the current frame. The rest of the display presents those data values graphically, or with greater resolution. In a graphical display, the marker consists of two vertical bars that enclose the information of the current frame.

Zoom In or Out

The graphical display shown in Figure 18.28 on page 469 lets you zoom in, to see more details, or zoom out, for a better general view. *Zoom in* and *Zoom out* are selections of the *Trace* popup menu.

Other ways to zoom in are choosing the **Trace>Zoom In** menu selection or pressing the 'I' key of the keyboard.

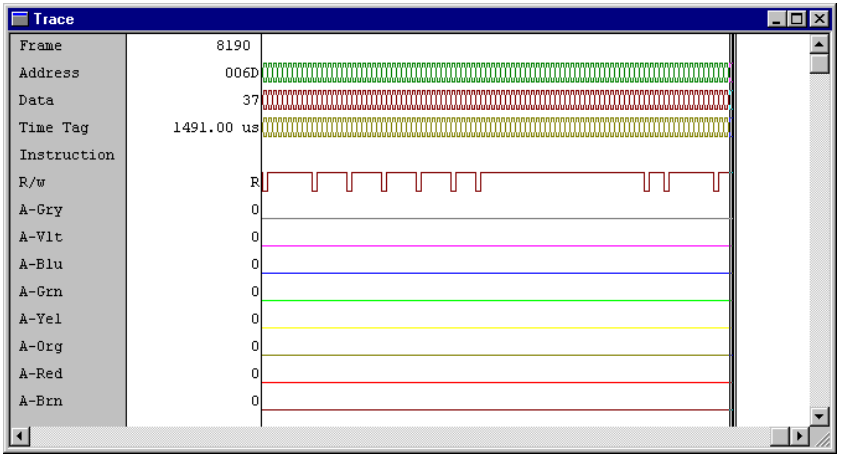
Figure 18.28 Trace Window - Zoom In Display



The figure above is a zoomed-in graphical display.

To zoom out, choose the **Trace>Zoom Out** menu selection, or press the *O* key of the keyboard. The Figure 18.29 on page 470 shown below is a zoomed-out graphical display.

Figure 18.29 Trace Window - Zoom Out Display



Dragging the marker over a bus analyzer display may also generate updates in component windows, such as the source and assembly windows. In other words, moving the marker over a bus analyzer textual or graphical display lets you view and thoroughly examine data in continually updated windows.

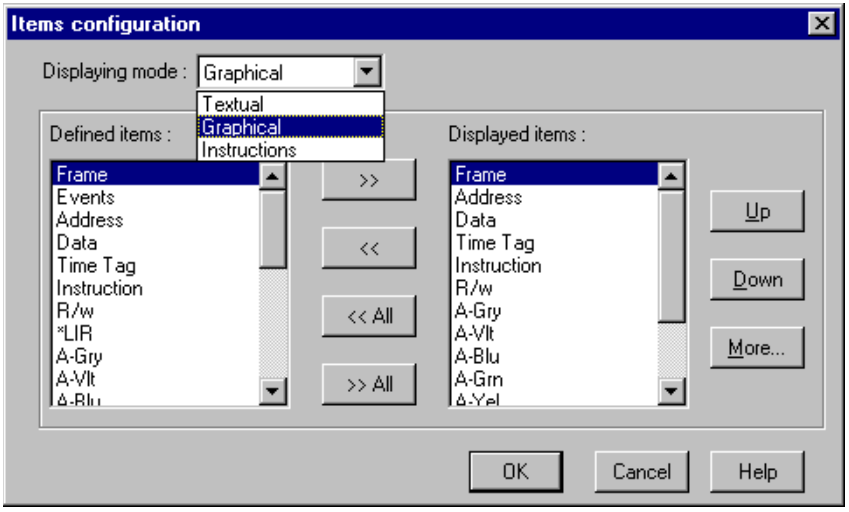
ShowLocation

To activate the *ShowLocation* selection for a frame, select the frame, click the right mouse button to activate the **Trace** menu, then select *ShowLocation*. Use a left mouse click to position the marker. This automatically updates the source and assembly windows.

Add / Remove Items in the *Trace* Window

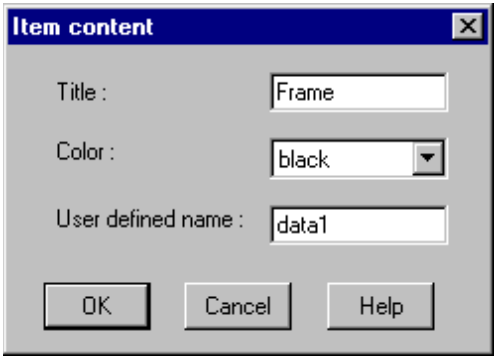
You can add new items to the *Trace* window. The default items are: *Frame*, *Events*, *Address*, *Data*, *Time Tag*, *Instruction* and *R/W* but you can add any other items of the *Items configuration* dialog box. This same dialog box shown in Figure 18.30 on page 471 lets you remove items from the *Trace* window. You can even drag items into the window.

Figure 18.30 Items Configuration Dialog Box



To edit an item, change its color, or assign a specific name, click the *More* button. This opens the *Item content* dialog box shown in Figure 18.31 on page 471:

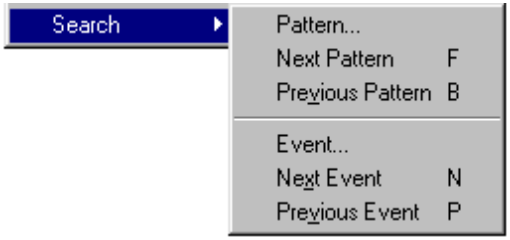
Figure 18.31 Item Content Dialog Box



Scrolling the Display

You can use display scrollbars as you would those of other Windows applications, or you can scroll to a specific trace buffer frame. It is also possible to search for one or more trigger events, or to search for a specific pattern, as the Figure 18.32 on page 472 illustration indicates.

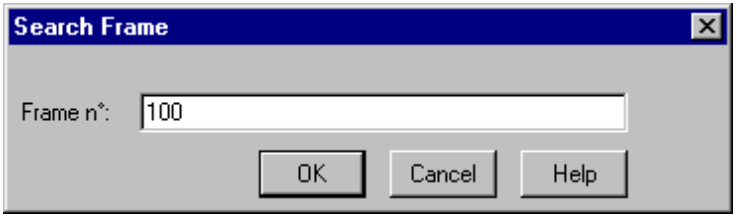
Figure 18.32 Search Menu



Search for Frame

To search for a frame, choose the **Trace>Go to Frame...** menu selection to open the dialog box shown in Figure 18.33 on page 472, then enter the frame number. The marker moves to the frame. If necessary, the system scrolls the bus analyzer window to make the frame visible. If the system cannot find the specified frame, an error message reports that fact.

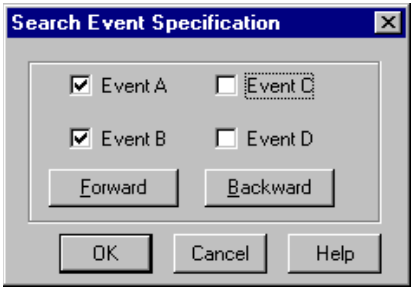
Figure 18.33 Search Frame Dialog Box



Search for Events

To search for an event occurrence, choose the **Trace>Search>Event...** menu selection. This opens the *Search Event Specification* dialog box as shown in Figure 18.34 on page 473. Check the checkboxes of one or more events.

Figure 18.34 Search Event Specification Dialog Box



OK Button

To store your specification of events (not searching immediately), click *OK*.

Forward

To search forward for the next frame that matches your event specification, click *Forward*.

Backward

To search backward for the next frame that matches your event specification, click *Backward*.

Next Event

To search for the next occurrence of the specified event, choose the **Trace>Search>Next Event...** menu selection. The analyzer searches forward from the selected frame to find the next frame that matches your event specification. (Pressing the keyboard *N* key is another way to activate this search.) If this search finds a matching frame, the marker moves to the frame. If necessary, the system scrolls the bus analyzer window to make the frame visible. If the search does not find a matching frame, an error message reports that fact.

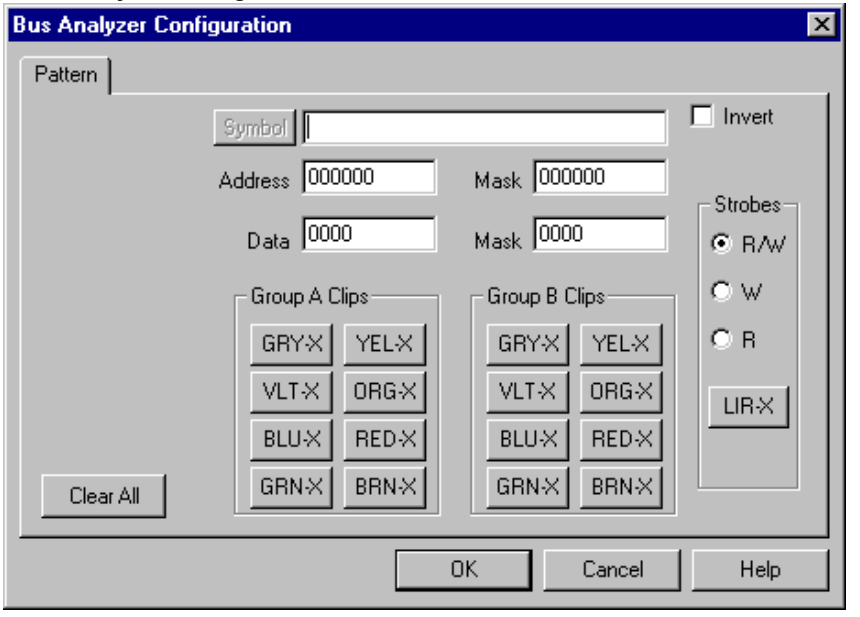
Previous Event

To search for the previous occurrence of the specified event, choose the **Trace>Search>Previous Event** menu selection. The analyzer searches backwards from the selected frame to find the previous frame that matches your event specification. (Pressing the keyboard P key, without activating the Trace menu, is another way to activate this search.) If this search finds a matching frame, the marker moves to the frame. If necessary, the system scrolls the bus analyzer window, to make the frame visible. If the search does not find a matching frame, an error message reports that fact.

Search for a Pattern

To search for a frame that matches a pattern, you must define the search pattern (an address, a data word, logic clips, and four miscellaneous signals). To do so, choose the **Trace>Search>Pattern...** menu selection. This opens the *Pattern* page of the *Bus Analyzer Configuration* dialog box as shown in Figure 18.35 on page 474.

Figure 18.35 Bus Analyzer Configuration Window - Pattern Tab



Use this tab to define the search pattern, in the same way you completed the Triggers tab.

The bus analyzer will look at specific collected cycles, match the frame that shows the pattern, then display this frame in the debugger window.

Address

In this edit box, specify the *Address* to be matched.

Data

In this edit box, specify the *Data* value to be matched.

Strobes

Use this area to specify the *R/W* and *LIR-X* states to be matched.

Group A & B Clips

Use these areas' toggle controls to specify each logic clip as *High (H)*, *Low (L)* or *Don't Care (X)*.

The clips buttons show the Group A and B logic clips with their respective colors.

You use logic clips to trace the signals of your debugger connection as it runs under software control. When a trigger occurs, a breakpoint shows you the states of significant logic signals before, at, and after the trigger. Such states can be a search pattern to be matched.

Invert

To specify matching any frame *without* the defined pattern, check the *Invert* checkbox.

OK Button

To save your pattern values and close the dialog box, click *OK*.

Cancel Button

To close the dialog box without changing any parameter values, click *Cancel*.

Next Pattern

To search for the next occurrence of the pattern, choose the **Trace>Search>Next Pattern** menu selection. If the search does not find a matching pattern, an error message reports that fact.

Previous Pattern

To search for the previous occurrence of the pattern, choose the **Trace>Search>Previous Pattern** menu selection. If the search does not find a matching pattern, an error message reports that fact.

Commands in the HI-WAVE Run Menu

These menu commands control the program flow.

Continue

Resumes execution of a halted program. Execution continues until it reaches a breakpoint, until the system detects a run time error, or until you enter a *Halt* command.

Halt

Stops the target processor, so that you can examine the state of the application program.

Step

Advances a halted application a single, source-level step. That is, execution continues to the next source reference.

Flat

Advances the application a single step. Same as Step, but does *not* step into called procedures.

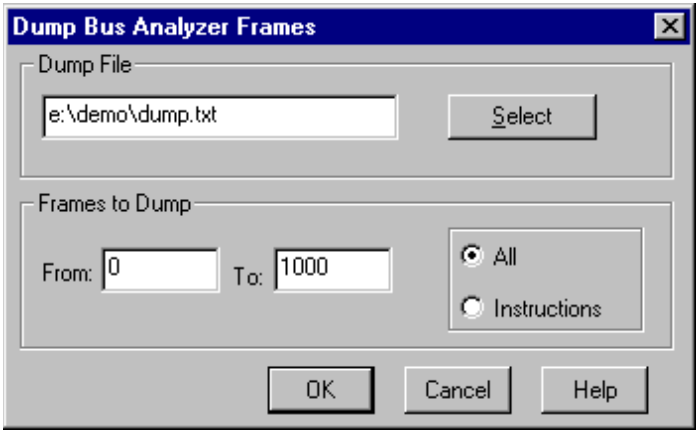
Asm Step

Advances a halted program one machine instruction. Same as the single step command in the assembly window.

Dumping Bus Analyzer Data to File

To dump the Bus analyzer data to a file, choose the **Trace>Dump...** menu selection. This opens the *Dump Bus Analyzer Frames* dialog box as shown in Figure 18.36 on page 476.

Figure 18.36 Dump Bus Analyzer Frames Dialog Box



Dump File

This dialog box lets you specify frames in the bus analyzer window, then dump those frames to a file.

In the *Dump File* edit box, enter the name of the file that will receive the dumped frames.

Select

Click the *Select* button to open a standard file-select dialog box. You can use this dialog box to specify the filename value for the *Dump File* edit box.

Frames to Dump

Use this area's edit boxes to define the range of frames to be dumped to the file.

All

To have the analyzer dump all frames of the range, select the *All* option button.

Instructions

To have the analyzer dump only frames in which an instruction starts (that is, an instruction also in the defined range), select the **Instructions** option button.

The trace window display only frame as shown in Figure 18.37 on page 477.

Figure 18.37 Trace Window - Analyzer Dump Only Frame

Trace

Frame	Events	Address	Data	Time Tag	Instruction	R/w	A-Gry	A-V
8172	----	0068	00	3670.50 us		R	0	0
8173	----	0468	B3	3671.00 us	CPX 0x70	R	0	0
8174	----	0469	70	3671.50 us	*	R	0	0
8175	----	0070	00	3672.00 us		R	0	0
8176	----	046A	25	3672.50 us	BCS *-44	R	0	0
8177	----	046B	D4	3673.00 us	*	R	0	0
8178	----	046B	D4	3673.50 us		R	0	0
8179	----	046C	26	3674.00 us	BNE *4	R	0	0
8180	----	046D	04	3674.50 us	*	R	0	0
8181	----	046D	04	3675.00 us		R	0	0
8182	----	046E	B1	3675.50 us	CMP 0x71	R	0	0
8183	----	046F	71	3676.00 us	*	R	0	0
8184	----	0071	16	3676.50 us		R	0	0
8185	----	0470	23	3677.00 us	BLS *-50	R	0	0
8186	----	0471	CE	3677.50 us	*	R	0	0
8187	----	0471	CE	3678.00 us		R	0	0
8188	----	0440	B6	3678.50 us	LDA 0x6F	R	0	0
8189	----	0441	6F	3679.00 us	*	R	0	0
8190	----	006F	08	3679.50 us		R	0	0

OK Button

Click the *OK* button to close the dialog box and dump the bus analyzer data to the file.

Cancel Button

Click the Cancel button to close the dialog box without dumping any data.

19

MMEVS0508 Emulator

MMEVS General Description

The M68MMEVS05/08 Motorola Modular Evaluation System (MMEVS) is Freescale's tool for developing embedded systems based on an MC68HC05 or MC68HC08 microcontroller unit (MCU).

A modular emulation system, the MMEVS provides interactive control of a microcontroller application when connected to your connection (target system).

A complete MMEVS includes:

- Platform board (M68MMPFB0508),
- Emulation module (EM),
- Target cable assembly.

The EM completes MMEVS functionality for a particular MCU or MCU family.

MMEVS Features

MMEVS features include:

- Real-time, non-intrusive, in-circuit emulation
- MC68HC11K1 system controller, for fast command transfer
- 64 kilobytes of emulation memory, to accommodate the largest available ROM size
- 64 hardware instruction breakpoints over the 64-kilobyte memory map
- A personality file for each EM. (Each personality file defines a foreground memory-map.)
- Latch-up resistant design (47-Ω series resistor on I/O connections to the target system) to make power-up sequencing unimportant.
- RS-232 operation speeds as high as 57600 baud
- Compact size: 8.25 inches deep, 10.5 inches wide

For connection instructions, configuration instructions, and other related information, see the installation section of this hardware addendum. For similar information with regard to EMs, see the corresponding EM user's manual.

MMEVS Components

The MMEVS components include:

- **Platform board:** the M68MMPFB0508 platform board.
- **9-lead RS-232 serial cable:** the cable that connects the platform board to the host computer RS-232 port.
- **Serial adapter:** a DB9M-to-DB25F RS-232 adapter, for use with a 25-pin host-computer serial port.

Separately purchased components are:

- **Emulation module (EM):** a printed circuit board that completes MMEVS functionality for one or more MCUs. The two DIN connectors on the bottom of the EM fit into connectors on the top of the platform board, providing power and signal connections. The EM also has a connector for the target cable. Each EM comes with its own user's manual.
- **Target cable:** a flat, flexible cable that connects the MMEVS to the debugger target system. The cable's emulator terminator mates to the EM target connectors; the cable's head terminator mates to a target head adapter. (A target cable and a target head adapter make up a target cable assembly.)
- **Target head adapter:** a target-cable adapter that plugs into the MCU socket of the debugger target system. (For some EMs, the target head adapter plugs into a surface-mount adapter.)
- User supplied components are:
  - **Host computer:**
  - **Power supply:** Required power is +5 volts @ 1 amp

MMEVS Installation

The complete MMEVS installation consists of:

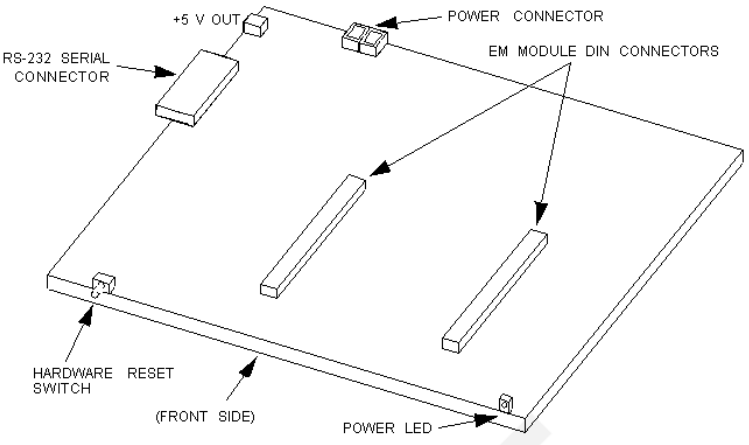
- on page 483Configuring the Platform Board on page 483,
- Installing the EM on page 485),
- Removing the EM on page 485, and
- Making Cable Connections on page 485.

This section explains all items except EM configuration. (As EM configuration is specific to each EM, you must follow the instructions of your EM user's manual.)

In addition, this section explains the reset switch, and provides pin assignments and signal descriptions for the platform-board serial connector.

Figure 19.1 on page 483 shows the M68MMPFB0508 platform board. The hardware reset switch and power LED are on the front of the platform board. The power connector is to the back and the 25-pin RS-232 serial connector is on the left. The circular, +5-volt out connector (P4), also on the left side of the platform board, is reserved for future features.

Figure 19.1 M68MMPFB0508 Platform Board



Configuring the Platform Board

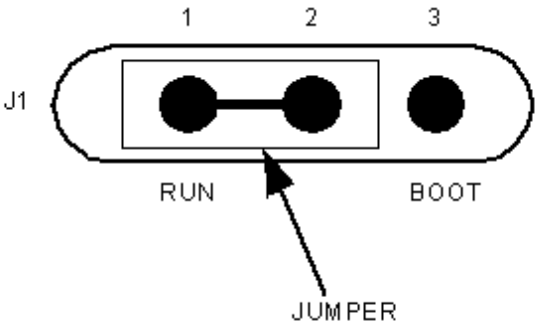
The MMEVS platform board has four jumper headers, all located near the front. Jumper header J1 is for factory test. Jumper headers J2, J3, and J4 control the voltage levels for ports A through D.

NOTE Before shipping platform boards, factory personnel configure them correctly for virtually all users. You should not reconfigure platform-board headers unless your EM user's manual tells you to.

Factory Test Header (J1)

The Figure 19.2 on page 484 shows the factory configuration of jumper header J1. The jumper between pins 1 and 2 is correct for MMEVS operation.

Figure 19.2 Factory Test Header (J1)



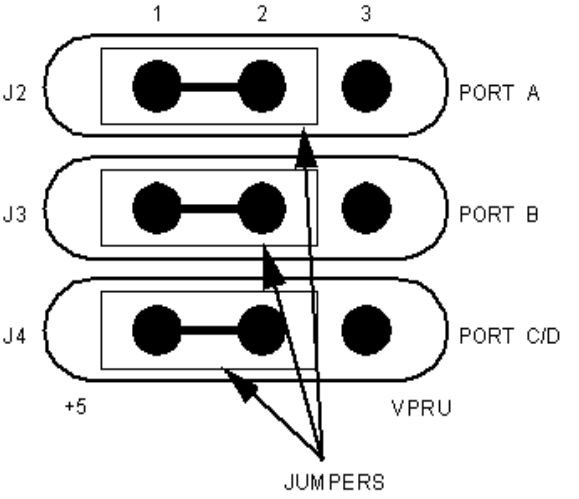
(Ignore the alternate jumper position, which is reserved for factory tests.)

Port Voltage Control Headers (J2–J4)

Jumper headers J2 through J4, near the right front corner of the platform board, set the voltage levels for ports A through D. Header J2 is the port A control; header J3 is the port B control; and header J4 is the control for port C or D (whichever pertains to your EM).

The Figure 19.3 on page 484 shows the factory configuration. The jumpers between pins 1 and 2 of these headers set the +5-volt level for all ports. This is the correct configuration for MMEVS operation, unless your EM user's manual says that your EM is a low-voltage board.

Figure 19.3 Port Voltage Control Headers (J2–J4)



If your EM can operate at low voltage, you can configure any of the ports for the low-voltage level. To do so, reposition the corresponding header's jumper to pins 2 and 3.

Installing the EM

CAUTION Be sure to turn off power to the MMEVS when you install or remove an EM. This prevents power surges that could damage MMEVS circuits.

To install an EM:

1. Make sure that platform board power is off.
2. Make sure that nylon spacers are in the correct positions for the emulation module (EM).
3. Install the EM on the platform board: Carefully fit the female 96-pin DIN connectors (on the bottom of the EM) onto the corresponding male DIN connectors (on the top of the platform board). Snap the EM onto the spacers and make sure that the DIN connectors are joined firmly.

NOTE Many EM boards have 64-pin female DIN connectors. Like their 96-pin counterparts, these smaller connectors also mate with the male DIN connectors of the platform board. The connector keys ensure proper alignment.

Removing the EM

To remove an EM:

1. Make sure that platform-board power is off.
2. Unsnap all nylon spacers from the edges of the EM.
3. Carefully lift the EM straight up, separating it from the platform board.

Making Cable Connections

You must connect the platform board to the host computer and to a power supply. According to your specific application, you also may need to connect the EM to your debugger target system.

Host Computer Connection

Connect the 9-lead serial cable between the platform-board serial connector (on the left side) and a serial port of the host computer.

A 9-to-25-pin adapter came with your MMEVS. If the host serial port is a 25-pin connector, use this adapter between the port connector and the serial cable.

Target Cable Connection

To connect the MMEVS to a target system, you must use a target cable assembly: a target cable and a target head.

CAUTION Press only on the rigid plastic terminators of the cable. Pressing on the flexible part of the cable can damage the cable.

Follow these instructions:

1. Make sure that platform-board power is off
2. Make sure that power is not applied to the target system
3. The EM has one or two target connectors located on its right side. Connect the target cable's emulator terminator to the EM target connectors.
4. Connect the target cable's head terminator to the appropriate target head adapter.

NOTE The EM user's manual should identify the target head adapter appropriate for your target system. Make sure that the adapter and the target cable mate correctly.

5. Plug the target head adapter into the MCU socket (or surface-mount adapter) of the target system.

Power Connection

The final MMEVS connection is to a +5 Vdc @ 1.0 amp power supply that you provide.

Lever terminal P3 is the MMEVS power connector. Contact 1 (black lever) is the ground. Contact 2 (red lever) is for V_{DD}, the +5 Vdc power. Use 20 or 22 AWG wire for power connections. For each wire, trim the insulation back a short distance from the end, lift the appropriate P3 lever, insert the bare wire into P3, and close the lever.

CAUTION Do not use wire larger than 20 AWG. Such wire could damage connector P3.

This completes the cable connections.

Reset Switch

RS-232 handshake signals control MMEVS resets. A reset initializes the control board from its startup point. If the host serial port does not implement handshaking, you must reset the MMEVS manually. Press gently to trip the switch.

RS-232 Serial Connector

This subsection provides pin assignments and signal descriptions for the platform-board serial connector. For similar information about EM connectors, see the corresponding EM user's manual.

The Figure 19.4 on page 487 shows pin numbering for the platform board's 25-pin serial connector. Table 19.1 on page 487 lists the signals transmitted on the 9-lead serial cable.

Figure 19.4 Serial Connector

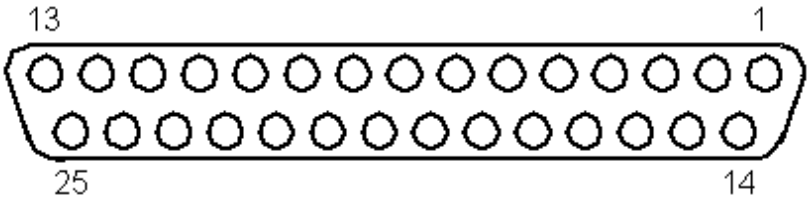


Table 19.1 Serial Connector and Cable Pin Assignments

Connector Pin	Mnemonic	Signal
1	GND	GROUND
2	TX	TRANSMIT DATA — Serial data input line
3	RX	RECEIVE DATA — Serial data output line
4	RTS	REQUEST TO SEND — Input signal that requests permission to transfer data
5	CTS	CLEAR TO SEND — Output signal that indicates a ready-to-transfer data status
6	DSR	DATA SET READY — Output signal that indicates on-line/in-line service/active status

Table 19.1 Serial Connector and Cable Pin Assignments (*continued*)

7	SIG-GND	SIGNAL GROUND — Signal ground or common return connection between the MMEVS and host computer
8	DCD	DATA CARRIER DETECT — Output signal that indicates detection of an acceptable carrier signal
9 — 19, 21 —25	—	No connection
20	DTR	DATA TERMINAL READY — Input signal that indicates on-line/in-line/active status

MMEVS Connection

An advanced feature of the debugger for the embedded systems development world is the ability to load different connection (targets), which implement the interface with target systems. This document introduces the MMEVS0508 Modular Evaluation System for the MC68HC08 MCU Family.

The MMEVS is a Freescale interface that the debugger uses to communicate with an external system (also called a connection or *target system*).

With this interface, you can download an executable program from the Debugger environment. The destination of this program is an external connection, based on a Freescale MCU, that executes the program. Debugger receives feedback of real connection behavior.

Debugger fully supervises and monitors the target MCU. That is, Debugger controls the CPU execution. You can read and write in internal or external memory (even when the CPU is running); you can single-step, run, or stop processes in the CPU.

NOTE As an external MCU executes the code, the MMEVS connection cannot provide memory statistics. This means that you cannot use the MMEVS connection for profiling, coverage analyzing, watchpoints, or I/O simulation.

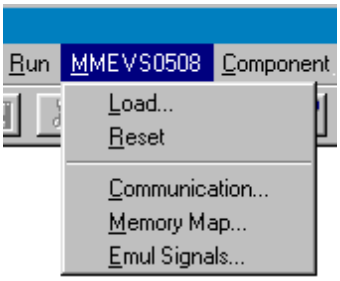
MMEVS0508

The MMEVS0508 is a Freescale emulator system for CPU05 or CPU08 MCUs that provides emulation memory.

MMEVS0508 Connection Menu

The functions above are specific for the MMEVS0508, so are available only with this emulator component. To use these features, select the corresponding menu commands, such as *MMEVS0508* | ..., as shown in Figure 19.5 on page 489.

Figure 19.5 MMEVS0508 Connection Menu



These functions are part of the MMEVS0508 Freescale Emulation System. Their supporting libraries consist of dialog boxes and routines to interface the debugger and the hardware.

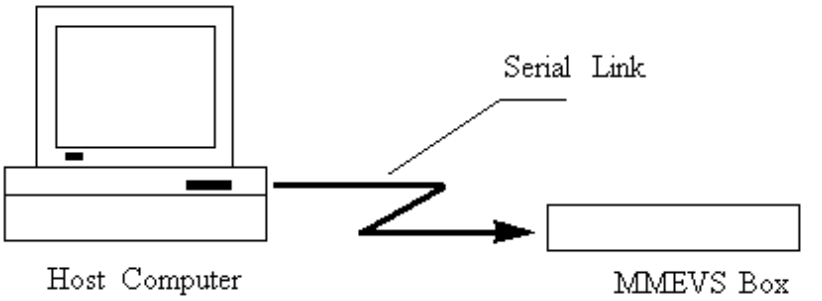
Interfacing Your System and Connection

The MMEVS0508 connects to an RS-232 serial port of your system.

Hardware Connection

Use the cables that came with the MMEVS0508 for the hardware connection (the Figure 19.6 on page 490 depicts this connection). Configure the host to be a data terminal, so that it sends data on the TxD lead and receives data on the RxD lead (as the MMEVS0508 hardware manual explains).

Figure 19.6 Hardware Connection

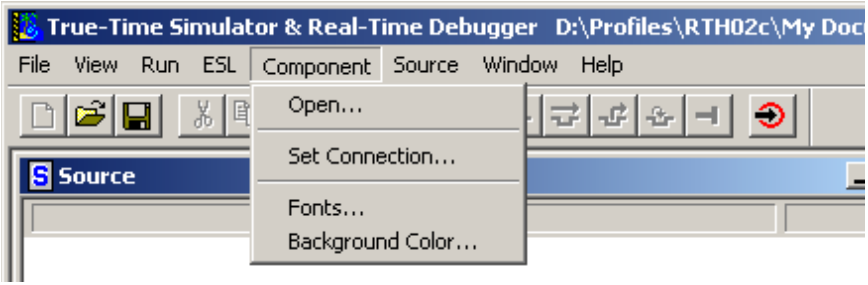


Loading the MMEVS0508 Connection

Usually, the `PROJECT.INI` file specifies the connection. To make the MMEVS the connection, change the file's "`Target=`" line to "`Target=Motosil`". The MotoSIL driver automatically detects the MMEVS connection to your system. However, if the driver detects nothing, an error message informs you that the connection is not connected to the expected port. Additionally, the *Communications Device Specification* dialog box appears, so that you can set the correct baud-rate and communication-port parameter values. See the *Communication Configuration* section of this document for more details.

Another way to load the MMEVS0508 connection is selecting *Set Connection...* from the *Component* Menu (See Figure 19.7 on page 490), then choosing **MotoSIL** from the list of possible connections.

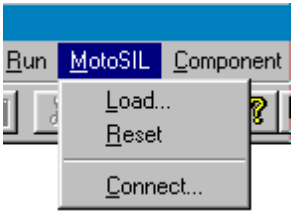
Figure 19.7 Component Menu - Set Connection Option



The MotoSIL driver automatically tries to find the MMEVS target, behaving as the text above explains.

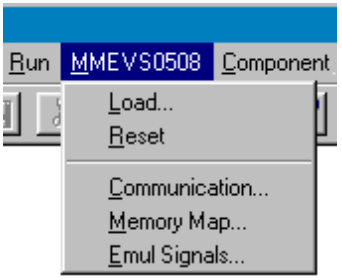
If MotoSIL does not detect a target, the *MotoSIL* menu remains in the main menu bar, as shown in Figure 19.8 on page 491:

Figure 19.8 Motosil Menu



But after successful target loading, the *MMEVS0508* menu replaces the *Target* or *MotoSIL* menu in the main menu bar, as the Figure 19.9 on page 491 depicts it.

Figure 19.9 MMEVS0508 Menu

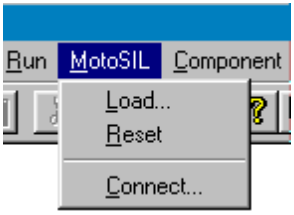


Communication Configuration

In most situations, Debugger uses its default values to set communication with the MMEVS automatically. In case of any problem, the dialog box below appears, so that you can correct settings.

NOTE Another way to open this dialog box is to select **MotoSIL | Connect...** from the menu bar (see Figure 19.10 on page 492). This method is appropriate if when previous connection attempts failed and Motosil still is in the main menu.

Figure 19.10 Changing the Connection Setting

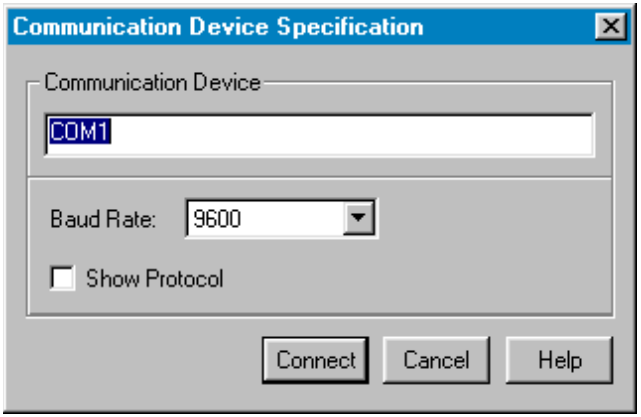


Make sure that host-computer parameter values are correct; make sure that the serial-communication setting is correct. Otherwise, communication between Debugger and the connection is not possible.

Communication Device Specification

If the host and target are not connected, or if the communication device is not properly indicated, the *Communications Device Specification* (Figure 19.11 on page 492) dialog box appears:

Figure 19.11 Communication Device Specification Dialog Box



Type the name of an available communication device in the *Communication Device* edit box, use the drop-down control to set the baud rate, then click *Connect*. (The default communication device is COM1.)

Once connection succeeds, Debugger saves the settings as defaults for later debug sessions. Should the connection fail, a message box so informs you, so that you can define a new communication device. To quit the dialog box and the environment, click *Cancel*.

NOTE Saving the communication device and the baud rate through this dialog box overrides environment variables `BAUDRATE` and `COMDEV` of the `default.env` file

Data Format

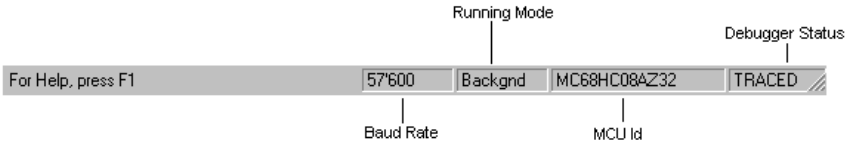
The MMEVS0508 data format is 8 data bits, 1 stop bit, no parity, and a variable baud rate. The default speed is 9600 baud, unless you change this default via the menu selection *MMEVS0508 | Communication...*.

Communication speeds of 1200 through 115200 baud are available, depending on the host-computer hardware.

MMEVS Debugger Status Bar

Once you have loaded the MMEVS target component, the Debugger status bar gives specific information, as explained by Figure 19.12 on page 493.

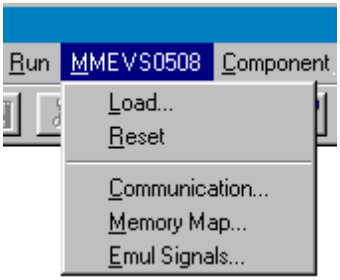
Figure 19.12 Status Bar



From left to right, this information is: the serial-communication baud rate, the Debugger running mode, the MCU name (depending on the MCU-Id), and the debugger status.

MMEVS0508 Menu Options

Figure 19.13 MMEVS0508 Connection Menu



Communication Baud Rate

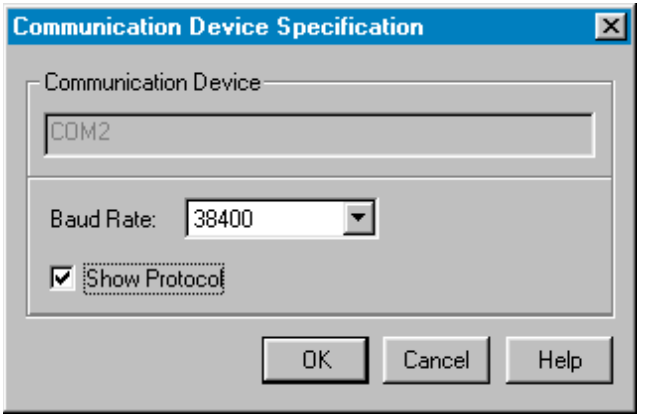
You should specify the baud rate for host-computer-to-MMEVS0508 communication early in a session. The system operates most efficiently at the maximum baud rate that the host computer supports.

You can modify this baud rate, as text below explains.

Communication

Select *MMEVS0508* | *Communication*... to display the *Communication Device Specification* dialog box, as shown in Figure 19.14 on page 494. Use the drop-down control to specify the maximum value (115200 baud), or to specify the maximum rate your host supports. If communication fails, Debugger reverts to the previous successful baud rate.

Figure 19.14 Communication Device Specification Dialog Box



Maximum Baud Rate

The maximum baud rate depends on the speed and interrupt load of the host computer. For slow notebook computers, or for computers running in a network, the maximum baud rate may be as low as 19200. A buffered I/O card may allow the maximum rate of 115200 for any host computer. The default value is 9600.

Show Protocol

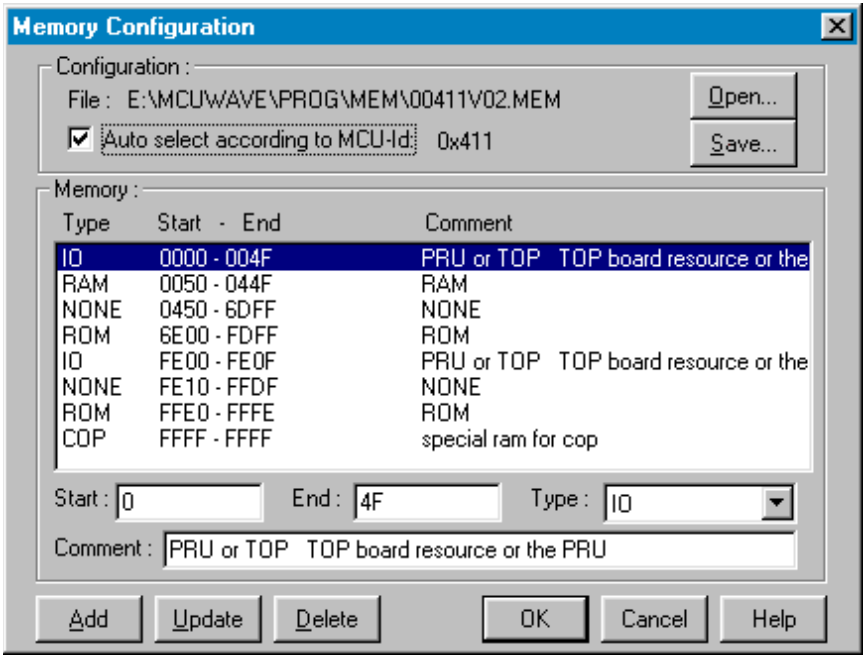
If you check the *Show Protocol* check box, the system reports all commands and responses in the command line window.

NOTE Support personnel use this feature.

Memory Configuration

To view the memory layout, choose *MMEVS0508* | *Memory Map...*. This opens the Memory Configuration dialog box, illustrated by Figure 19.15 on page 495

Figure 19.15 Memory Configuration Dialog Box



This dialog box shows the target's memory setup. The system automatically loads this setup if you check the *Auto select according to MCU-Id* check box. Debugger identifies and sets the memory map through the processor MCU-Id. To open another configuration, click the *Open* button. To save modifications to the current configuration, click the *Save...* button.

Personality (.MEM) Files

For proper operation, the Motosil target must load the personality file (`.MEM` file) that matches the connected Emulation Module (EM).

The `.MEM` file filename format is:

```
0nnnnVxx.MEM
```

where `nnnn` is the four-digit, hexadecimal `MCU-Id` number of the MCU, and `xx` is a two-digit version number.

If that target cannot find this file, or if the file is not valid, the *Error* message box shown in Figure 19.16 on page 496 appears.

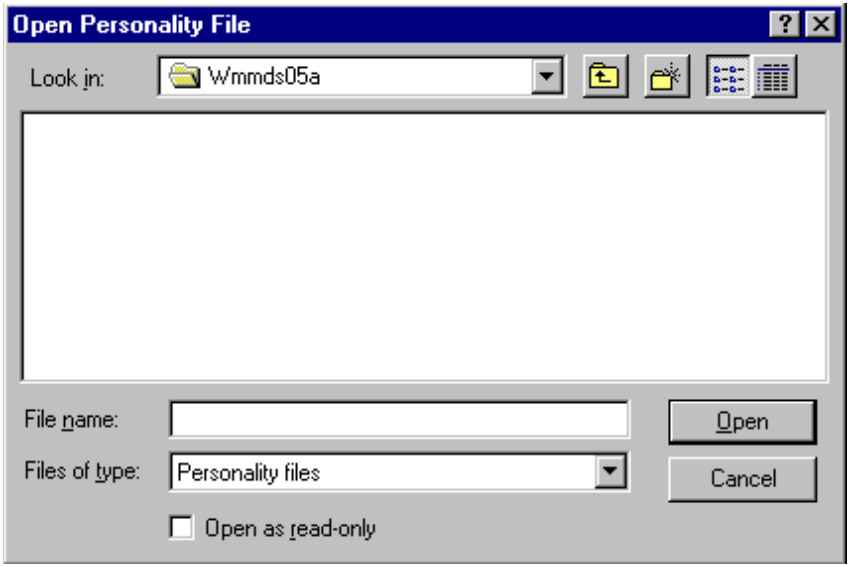
Figure 19.16 Personality File Not Found Error Message Box

Error
Personality file for Id 'A18' was not found in directory 'C: \MCU08\PROG\MEM'. If a matching .MEM file is available click 'Retry' to select and install this file in the directory 'C:\MCU08\PROG\MEM'. If such a file is not available, proper operation will not be possible.
Retry
Cancel

Click *Cancel* to open the *Communication Device Specification* dialog box, instead of establishing the connection.

Click *Retry* to bring up the *Open Personality File* dialog box, represented in Figure 19.17 on page 497

Figure 19.17 Open Personality File Dialog Box



This dialog box lets you browse to find and open the necessary `.MEM` file.

If you select another invalid `.MEM` file, the error message box and the *Communication Device Specification* dialog box reappear.

If the `.MEM` file is valid, the target loads the file, copying it into the `\PROG\MEM` directory, with '`V00`' as the version number (for example, `00A18V00.MEM`).

Note that the *Memory Configuration* dialog box displays the current memory map.

When starting Debugger:

If you have checked the *Auto Select according MCU-ID* checkbox, the system automatically loads the default personality file for the MCU-ID. If this checkbox is clear, the system automatically loads the most recently opened or saved memory map file.

You can use the *Memory Configuration* dialog box to modify the memory configuration, then save this new configuration into a memory configuration file. (Click the *Save* button.)

You can use the *Memory Configuration* dialog box to load a different memory configuration file (one that you previously defined and saved). (Click the *Open* button.)

Another way to load personality files (or any memory configuration files) is to enter the `LOADMAP` command in the command line.

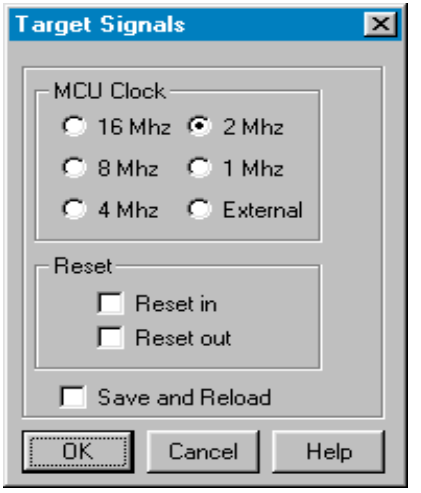
Memory

The *Memory* area lets you specify the “Real-Time Memory”. For the MMEVS0508, this is dual-ported memory that you can assigned to any valid RAM or ROM address. While the MMEVS0508 is running, Debugger can display and modify this “Real-Time Memory”. However, if a portion of this memory overlays internal MCU I/O, RAM, or EEPROM, the *Memory Configuration* dialog box can only display, not monitor, that memory portion.

Signals Emulation

To specify MMEVS0508 emulator signals, choose the *MMEVS0508 | Emul Signals...* menu selection. This opens the Target Signals dialog box, depicted in Figure 19.18 on page 498.

Figure 19.18 Target Signals Dialog Box



The Target Signals dialog box lets you specify the MCU clock and the reset signal connection.

CAUTION In order to use any of these clock signals, you must configure EM jumper headers correctly.

MCU Clock

The *MCU Clock* area lets you specify a different MCU clock, provided that the EM configuration is correct.

Reset

The *Reset* area lets you specify the reset-signal connection with the target system.

The *Target Signals* dialog box displays settings that the system reads from the MMEVS0508. Click *Ok* to close the dialog box, and to write values back to the MMEVS0508. Check the *Save and Reload* checkbox to have the system save the configuration, then reload this configuration the next time you start the Debugger debugger.

Default Connection Setup

As with any connection, you can use the *Component* menu to load the MMEVS connection, or you can set the MMEVS connection as a default in the `PROJECT.INI` file. This file should be in the project directory.

Example `PROJECT.INI` file:

```
[DEFAULTS]
Window0=Source      0    0   50   40
Window1=Assembly   50    0   50   40
Window2=Register   50   40   50   30
Window3=Memory     50   70   50   30
Window4=Data        0   40   50   25
Window5=Command     0   65   50   20
Window6=Module      0   85   50   15
Target=Motosil

[Motorola ESL]
COMDEV=COM2
BAUDRATE=57600
SHOWPROT=1
```

NOTE For more information about the `PROJECT.INI` file, see the debugger manual.

Freescale ESL Parameters

In normal use, you set these parameters in the `PROJECT.INI` file once, interactively, during installation. You use these parameter values in subsequent debugging sessions.

COMDEV

This parameter specifies the host-computer communication port. COM1 is the default communication device for PCs. The default for UNIX systems is `/dev/ttya`.

For a PC: Any valid communication device (`COM1`, `COM2`, etc.).

Example: `COMDEV=COM2`

For SUN:Any valid communication device (`/dev/ttya`, etc.).

Example: `comdev=/dev/ttyb`

BAUDRATE

This parameter specifies the communication baud rate between the host computer and the target system. The Debugger default is 9600 baud, but you may set any of these baud rates:

`1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200`

Example:`BAUDRAUTE=19200`

SHOWPROT

This parameter specifies whether to report commands and responses in the command line window. To have the system report the commands and responses, give this parameter the value 1. To have the system *not* do this reporting, give this parameter the value 0. (Another way to specify reporting is checking the *Show Protocol* checkbox of the *Communication Device Specification* dialog box.)

Please see the section *Communication Configuration, Communication Device Specification.*

Book III - HCS08 Debug Connections

Book III Contents

Each section of the Debugger manual includes information to help you become more familiar with the Debugger, to use all its functions and help you understand how to use the environment. This book, the HCS08 Debugger Connections, defines the connections available for debugging code written for HCS08 CPUs.

This book consists of the following sections:

- "HCS08 Full Chip Simulation" on page 503
- "HCS08 P&E Multilink/Cyclone PRO Connection" on page 537
- "Softec HCS08 Connection" on page 547
- "HCS08 Serial Monitor Connection" on page 555
- "HCS08 On-chip DBG Module" on page 567

20

HCS08 Full Chip Simulation

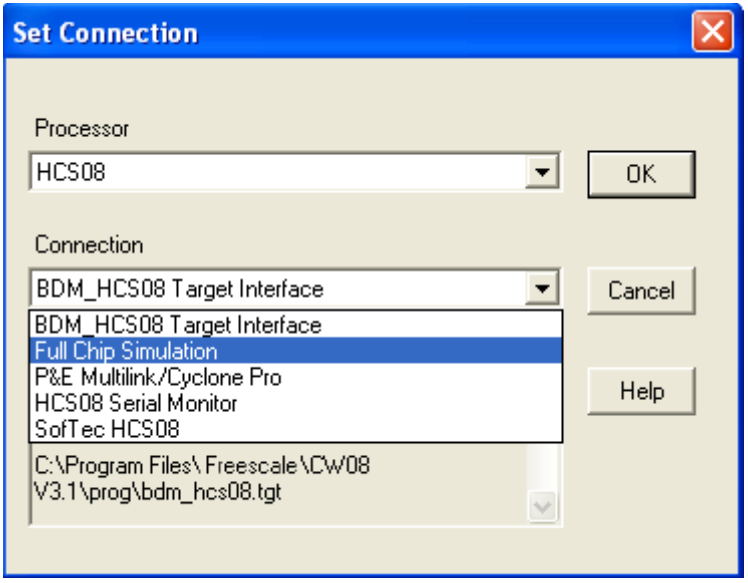
Full Chip Simulation (FCS) does not involve real input and output. Because of this, it does not require a target device to be connected to your PC. The HCS08FCS connection simulates the execution of code on the user's MCU system, including the function of any peripherals associated with the device that you select. For more detailed information, refer to the Full Chip Simulation description for the module that you are using.

Configuration Procedure

To select Full Chip Simulation as the debugger connection:

1. Choose the Full Chip Simulation option from the set connection dialog box. See Figure 20.1 on page 503.
2. Click the OK button.

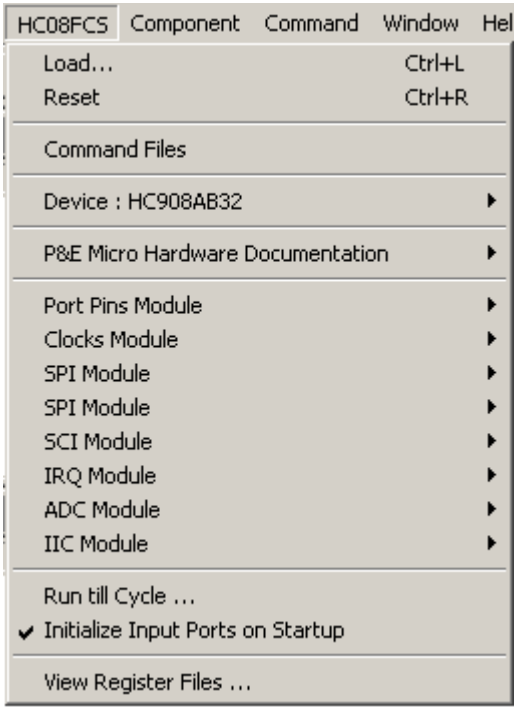
Figure 20.1 Set Connection Dialog Box



Connection (HCS08FCS) Menu

Once you have chosen Full Chip Simulation as your debugger connection, the name of the Connection menu is updated and addition options are added.

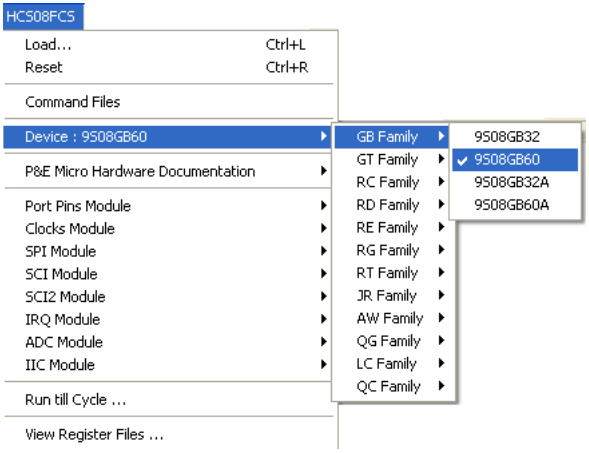
Figure 20.2 HC08FCS Menu



Device Option

The Device selection of the HCS08FCS menu allows the user to select the particular Freescale processor that they wish to use. When choosing the Device option from the HCS08FCS menu, extended menus will open which allow you to select the family (e.g. GB Family), and device type (e.g. 9S08GB60) of the MCU that you are using.

Figure 20.3 HCS08FCS Device Exteded Menus



Full Chip Simulation Module Commands

The HC08FCS Menu contains the Full Chip Simulation commands for the modules that have specialty commands associated with them for a chosen device. For more information about specific module commands please refer to the Full Chip Simulation section describing the module.

Run Till Cycle Command

The Run Till Cycle command will begin execution of code, and stop execution when the specified cycle count is reached. Please note that the parameter given is not the number of cycles that will be executed, but rather the total cycle count of the simulator (displayed in the Register Window).

This command is extremely useful for verifying specific timings of a given event; running until a given event is complete, or just before it completes to enable stepping through the event itself; or any application where cycle-timed execution is desired.

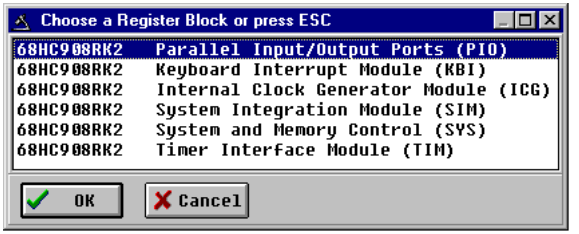
Initialize Input Ports On Startup

The Initiailze Input Ports on Startup option will initialize all simulated inputs to "$00" when the software is started, or when the Device Mode or Debug Target is switched. This initialization doesn't apply to a **reset** command. When this option is changed, the new state will take effect the next time HiWave is started (or Device Mode/Debug Target is switched).

View Register Files Command

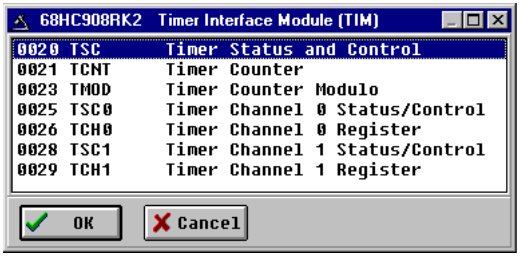
The “View Register Files...” selection in the HCS08FCS menu also gives the user the option of running the register file viewer/editor. If register files are available for the device that you have chosen, the “Choose a Register Block” window (see Figure 20.4 on page 506) will be opened. You may also open it by entering the R command in the Command Window command line.

Figure 20.4 Choose A Register Block Dialog Box


Choose a Register Block or press ESC
68HC908RK2 Parallel Input/Output Ports (PIO)
68HC908RK2 Keyboard Interrupt Module (KBI)
68HC908RK2 Internal Clock Generator Module (ICG)
68HC908RK2 System Integration Module (SIM)
68HC908RK2 System and Memory Control (SYS)
68HC908RK2 Timer Interface Module (TIM)
OK
Cancel

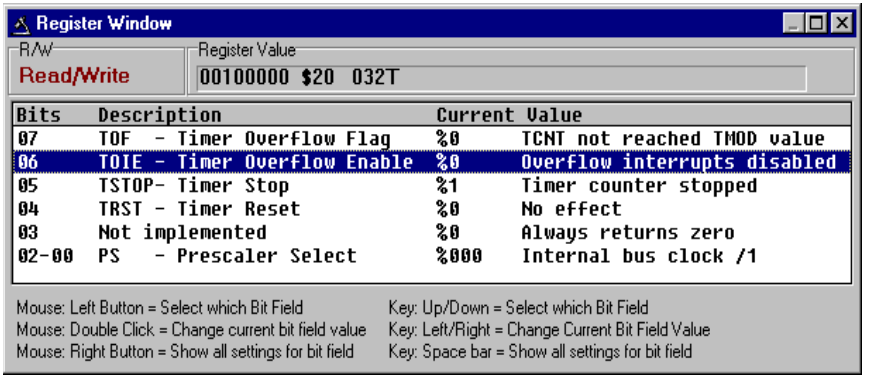
If register files have been installed on the host computer, selecting a block brings up the Register Block register listing (see Figure 20.5 on page 506), which shows a list of the files, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.

Figure 20.5 Timer Interface Module Register Listing


68HC908RK2 Timer Interface Module (TIM)
0020 TSC Timer Status and Control
0021 TCNT Timer Counter
0023 TMOD Timer Counter Modulo
0025 TSC0 Timer Channel 0 Status/Control
0026 TCH0 Timer Channel 0 Register
0028 TSC1 Timer Channel 1 Status/Control
0029 TCH1 Timer Channel 1 Register
OK
Cancel

Selecting a file brings up the Register Window (see Figure 20.6 on page 507), which displays the values and significance for each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.

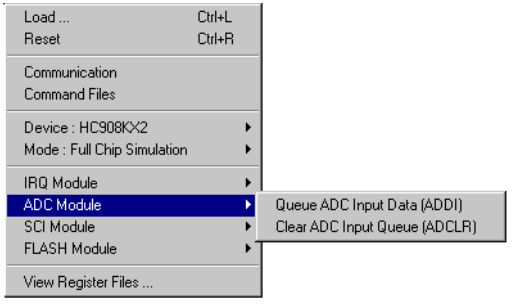
Figure 20.6 Register Window


Register Window
R/W
Read/Write
Register Value
00100000 $20 032T
Bits Description Current Value
07 TOF - Timer Overflow Flag %0 TCNT not reached TMOD value
06 TOIE - Timer Overflow Enable %0 Overflow interrupts disabled
05 TSTOP- Timer Stop %1 Timer counter stopped
04 TRST - Timer Reset %0 No effect
03 Not implemented %0 Always returns zero
02-00 PS - Prescaler Select %000 Internal bus clock /1
Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

Peripheral Modules Commands

If you have selected a device (see Device Option), the HCS08FCS Menu will display a list of peripheral modules and the associated commands for the device you have chosen.

Figure 20.7 HCS08FCS Menu: Peripherals/Commands Extended Menus



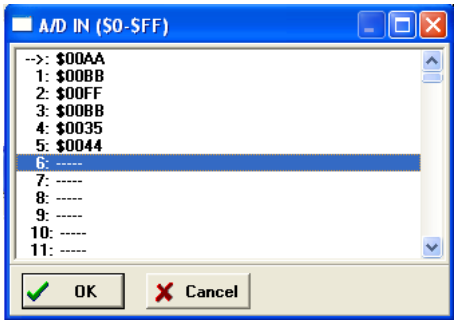
Placing your mouse over a peripheral opens a box which lists its associated commands. Click on a command in order to execute that command.

ADC Module

In Full Chip Simulation, this module will simulate all functionality of the Analog to Digital Conversion (ADC) module including data input on all ADC channels, flag polling, interrupt operation as well as the bus and CGMXCLK reference clock sources. Full Chip Simulation (FCS) mode uses the buffered input structure to simulate the ADC inputs. The user can queue up to 256 data values. To queue the ADC Input Data, the user should use

ADDI command in the command prompt. If the data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is provided, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the ADC. The conversion takes place after a proper value is written to the ADC Status and Control register. Once the conversion occurs, the arrow moves to the next value in the ADC Buffer.

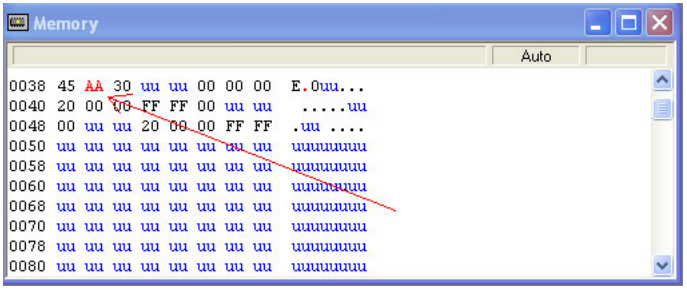
Figure 20.8 ADC IN Buffer Display



At any point, the ADCLR command can be used to flush the input buffer for the ADC simulation.

After the conversion is complete, the first queued value is passed from the data buffer into the ADC data register. It can be observed in the memory window by displaying the memory location corresponding to the ADC data register.

Figure 20.9 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

When the conversion is complete, the appropriate flag will be set. If interrupts are enabled, the Program Counter will change flow to the interrupt routine (as defined in the vector space of the MCU). For more information on ADC configuration, please refer to the Freescale user manual corresponding to the microprocessor that you are using.

ADC Module Commands

ADDI Command

The ADDI command allows the user to input the data into the ADC converter. If a data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next to the ADC. The maximum number of input values is 256 bytes.

Syntax:

>ADDI [<n>]

Where:

<n> The value to be entered into the next location in the input buffer.

Example:

>ADDI $55

Set the next input value to the ADDI to $55

>ADDI

Pull up the data window with all the input values.

ADCLR

The ADCLR command can be used to flush the input buffer for ADC simulation. This will reset the input data buffer and clear out all values. Notice that if the ADC is currently using a value, this command will not prevent the ADC from using it. See ADDI Command for information on how to access the input buffer of the ADC interface.

Syntax:

>ADCLR

Example:

>ADCLR

Clear the input buffer for ADC simulation.

Clock Generation Module

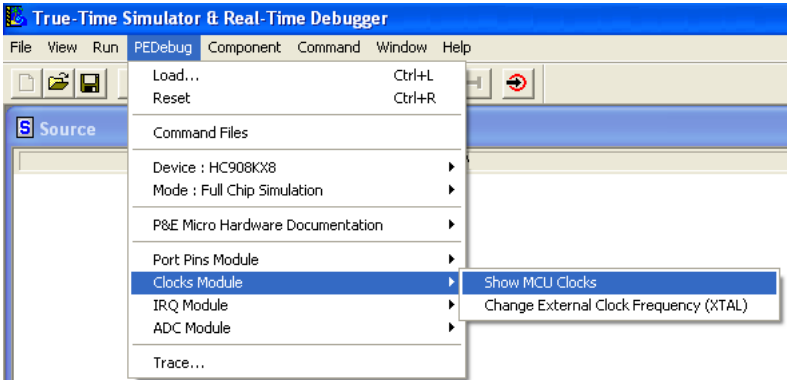
In Full Chip Simulation, this module will simulate all functionality of the Clock Generation Module (ICG), including:

- Phase Locked Loop (PLL) generation

- Automatic lock detection
- Interrupt
- Acquisition
- Tracking
- Flag polling

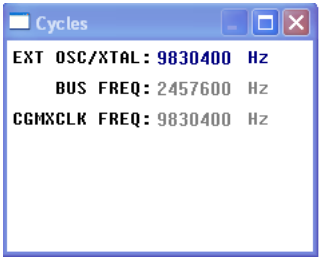
Full Chip Simulation mode uses simulated External Oscillator Frequency change command (XTAL) to allow the user to input the desired XTAL value. To check the current value of the External Oscillator, Bus Frequency and CGMXCLK Frequency, the user should open the HCS08FCS menu, then select Clocks Module -> Show MCU Clocks.

Figure 20.10 Clocks Module Extended Menu



Once the MCU Clocks Menu is selected, the Cycles Window containing all of the above-mentioned Clock Frequencies will be displayed.

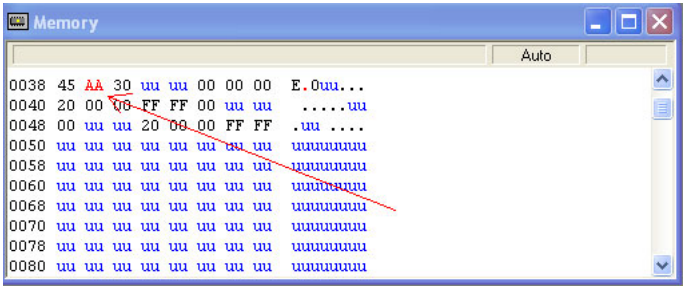
Figure 20.11 Frequency Display



Once the ICG is properly configured, the user can monitor the status of the PLL by polling the corresponding flag. If PLL interrupt is enabled, Full Chip Simulation will jump to an appropriate subroutine as long as the interrupt vector is properly defined. To observe the

flag going up as a result of the corresponding CPU event, situate your Memory Window on the memory location of the ICG Status and Control register.

Figure 20.12 Memory Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

For more information on how to properly configure Clock Generation, please refer to the Freescale reference manual corresponding to the microprocessor that you are using.

Clock Generation Module Commands

XTAL Command

The XTAL command is used to change the value of the simulated external oscillator. This in turn will affect the input to the PLL/DCO, and therefore the bus frequency. The P&E simulator is a cycle-based simulator, so changing the XTAL value will not affect the speed of simulation; it will, however, affect the ratio in which peripherals receive cycles. Certain peripherals which run directly from the XTAL will run at different speeds than those that run from the bus clock.

Syntax:

>XTAL <n>

Where:

n, by default, is a hexadecimal number, representing the simulated frequency of an external oscillator. Adding the suffix “t” to the n parameter will force the input value to be interpreted as base 10.

Example:

>XTAL

Brings up an input window. The default base for this input value is 10. However, this value can be forced to a hexadecimal format through use of the suffix “h”.

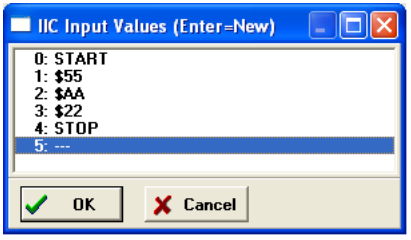
Inter-Integrated Circuit Module

In Full Chip Simulation, this module will simulate all functionality of the Inter-Integrated Circuit (IIC) module including:

- Flag polling
- Interrupt enabled mode
- Transmission and reception of external data
- Master and slave modes of operation
- START and STOP signal generation detection
- Acknowledge bit generation detection

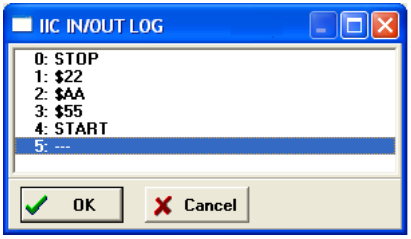
The Full Chip Simulation (FCS) mode uses the buffered input/output structure to simulate IIC inputs. The user can queue up to 256 data bytes into the input buffer. The output buffer of the USB module can also hold 256 output bytes. To queue the IIC Input Packets, one should use IICDI <...> command in the command prompt. For a more detailed description of the command, please refer to the IIC Commands section. If the IIC packet parameters are properly defined, the packet is placed into the next slot in the input buffer. Otherwise, if no parameters are provided, an IIC Input Buffer window is displayed. Different IIC packet parameters can be entered while the window is open including START, STOP, ACK, NACK and data bytes. An arrow points to the byte that will be used next as input to the IIC. The data from the IIC input buffer is written to the IIC module registers once the IIC module is turned on and properly configured for receiving data from an external IIC device. Once the simulation of the data transmission is over, the arrow moves to the next value in the IIC Input Buffer.

Figure 20.13 IIC Input Buffer Display



The IIC data input/output log buffer simulation allows the user to gain access to the past 256 IIC data bytes that have been shifted in and out of the module. To bring up the IIC IN/OUT LOG buffer dialog, the IICDO command should be used.

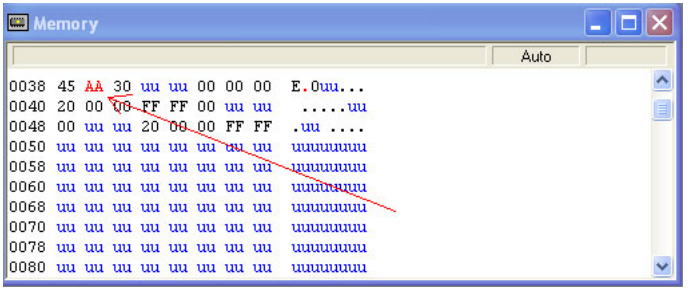
Figure 20.14 IIC IN/OUT LOG Buffer Display



At any point, IICCLR command can flush the input as well as input/output log IIC buffers.

After the IIC simulated input is received, the first queued in data byte is passed from the data buffer into the corresponding IIC module registers. It can be observed in the Memory Window by displaying the appropriate register location there.

Figure 20.15 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

The user can also observe different IIC flags in the Memory Window. If the module is run in Flag Polling mode, the user should poll the flag corresponding to the expected IIC event. If the IIC interrupts are enabled, the FCS will jump to an appropriate subroutine as long as the IIC interrupt vectors are properly defined.

For more information on how to configure IIC module for desired operation, refer to the Freescale user manual corresponding to the microprocessor that you are using.

Inter-Integrated Circuit Module Commands

IICDI Command

The IICDI command allows the user to input data into a buffer of data which will be shifted into the IIC module when it receives data from an external device. If a data parameter is given, the value is placed into the next slot in the input buffer. Otherwise if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. The maximum number of input values is 256.

This command is useful for either inputting response data from a slave target or for inputting data packets from an external master. Note that when the microprocessor attempts to read an acknowledge from an external device, and the next value in the buffer is neither ACK nor NACK, the microprocessor automatically receives an ACK signal (i.e. assumes ACK unless NACK is specified).

Syntax:

>IICDI [<n>][START][STOP][ACK][NACK]

Where:

<n> The value to be entered into the next location in the input buffer

START indicates the incoming START signal

STOP indicates the incoming STOP signal

ACK corresponds to ACK signal

NACK corresponds to NACK signal

For a detailed description of the IIC protocol and a proper way IIC module should be configured, please refer to the Freescale user manual corresponding to the Freescale microprocessor that you are using.

Example:

>IICDI

Pulls up the data window with all the input values

>IICDI 22 33

This is an example of data being returned from a slave device. Once the MCU transmits a start signal and the target address, it receives and ACK from the slave device. An ACK is implied unless a NACK is specified via the IICDI command. The next tow data bytes read will be 22 and 23. If the microprocessor attempts to read another byte, it will get an $FF value followed by a NACK signal (NACK because nothing remains in the input buffer). The receiving device will then generate a STOP signal. A more exact input from a device designed to return two bytes is:

>IICDI ACK 22 ACK 23 NACK

IIC in master mode transmits to a slave:

If the slave device acknowledges all output bytes of the transmitting device, there is not need to specify an input packet. If the master device were going to transmit an address and two bytes, the following packet would be equivalent to no packet:

>IICDI ACK ACK ACK

If, however, the slave receiver is designed to generate a NACK signal after the second received data byte, the proper response packet is:

>IICDI ACK ACK NACK

The address result being the first ACK, the first data result being the second ACK, and the second data byte being the NACK.

IIC in MASTER mode is not acknowledged by any Slave:

>IICDI NACK

If the NACK signal is entered before our master device transmits a START signal, then the master device will get back a NACK when it tries to read an acknowledge after the address is outputted. The master device will then generate a STOP signal and release the BUS.

IIC in SLAVE mode receives a Write from an external Master:

This example is for an external master which is writing to the microprocessor configured to simulate the slave mode operation. The packet contains both START and STOP signals which will put the simulated device into the slave mode.

>IICDI START 55 AA 22 STOP

This input should add 5 values to the input queue which is a packet from an external master including the following procedure steps:

1. A start signal comes in
2. The address $55 comes in specifying a write (slave receive); The current simulated device should have previously set its Address Register to $55
3. The data byte $AA comes in
4. The data byte $22 comes in
5. A STOP signal comes in

IICDO Command

The IICDO command displays a window, which shows data shifted in as well as shifted out of the IIC peripheral. An arrow is used to point to the last output value transmitted/ received. The maximum number of output values that the buffer can hold is 256.

Syntax:

>IICDO

Example:

IICDO

View data from the input/output log buffer for IIC simulation.

IICCLR Command

The IICCLR command can be used to flush the input and output buffers for IIC simulation. This will reset the buffers and clear all values. Notice that if the IIC is currently shifting a value, this command will not prevent the IIC from finishing the transfer.

Syntax:

>IICCLR

Example:

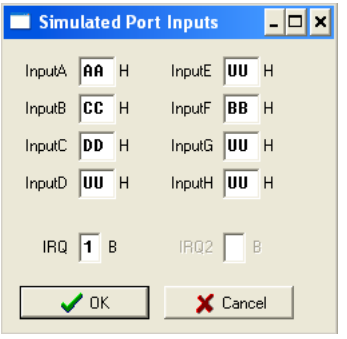
>IICCLR

Clear input and output buffers for IIC simulation.

Input/Output (I/O) Ports Module

In Full Chip Simulation, this module will simulate all input and output functionality of the Input/Output (I/O) Ports module. The Full Chip Simulation (FCS) mode uses a set of designated commands to simulate the input and output activity on corresponding I/O port pins. To define an input state of the specific port one should write the INPUT <x> <n> command in the Command line window. The <x> represents corresponding I/O port, while the <n> stands for the input value that should be written to this port. At the same time, the INPUTS command can be used to bring up the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports. Please see the documentation for the INPUT<x> Command and INPUTS Command for more information about the various forms of this command.

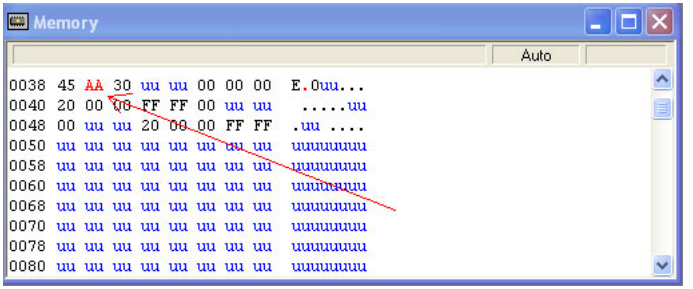
Figure 20.16 Simulated Port Inputs Dialog Box



Using the Simulated Port Inputs Dialog, the input value to any I/O port can be reconfigured. The INPUTS command can be used to reconfigure the output values on any

relevant I/O port. The manipulation of I/O port pins can be observed in the Memory Window.

Figure 20.17 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

Please note that if the regular I/O pins are multiplexed to be used by a different MCU Module, they might not be available for general I/O functionality. For more information on how to properly configure I/O pins, please refer to the Freescale user manual corresponding to the microprocessor that you are using.

Input/Output Ports User Commands

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax:

>INPUT<x> <n>

Where:

<x> is the letter representing corresponding port

<n> Eight-bit simulated value for port <x>

Example:

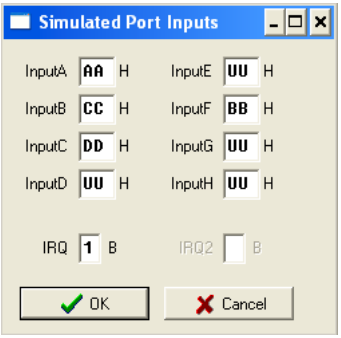
>INPUTA AA

Simulate the input AA on port A.

INPUTS Command

In Full Chip Simulation and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 20.18 on page 518. The use may then use this box to specify the input states of port pins and IRQ.

Figure 20.18 Simulated Port Inputs Dialog Box



When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax:

>INPUTS

Example:

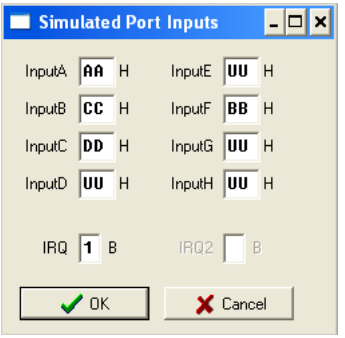
>INPUTS

Show I/O port input values.

External Interrupt (IRQ) Module

In Full Chip Simulation, this module will simulate the input, flag polling and interrupt functionality of the External Interrupt (IRQ) module. The Full Chip Simulation Mode uses the INPUTS command to let the user monitor and change the simulated value of the IRQ input pin state. Once the user enters the INPUTS command into the command line prompt, the Simulated Port Inputs window will come up. Please see the documentation for **INPUT<x> Command** and INPUTS Command for more information about the various forms of this command. In addition, the state of the IRQ pin can be modified directly using the IRQ <n> command (documented below).

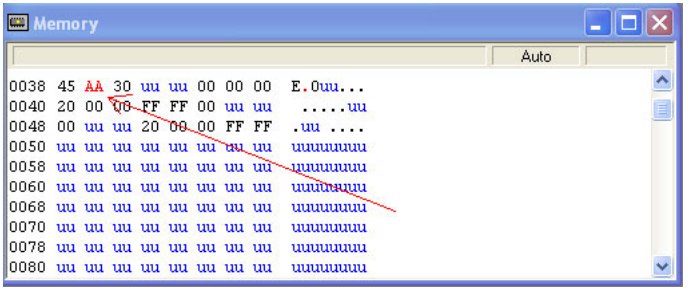
Figure 20.19 Simulated Port Inputs Dialog Box



After the IRQ event occurs, the appropriate flag will be set in the corresponding IRQ register. The user is able to poll the IRQ flag if the Polling Mode is simulated. In the Interrupt Mode, the simulator will branch to an appropriate interrupt subroutine as long as the IRQ interrupt vector is properly configured. For more information on IRQ configuration, please refer to the Freescale user manual corresponding to the microprocessor that you are using.

Following the IRQ event, the IRQ Flag can be observed going up in the IRQ Status and Control register.

Figure 20.20 Memory Component Window

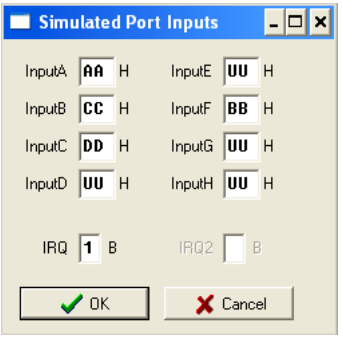

Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

IRQ Commands

INPUTS Command

In Full Chip Simulation and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 20.21 on page 520. The user may then use this box to specify the input states of port pins and IRQ.

Figure 20.21 Simulated Port Inputs Dialog Box



When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax:

>INPUTS

Example:

>INPUTS

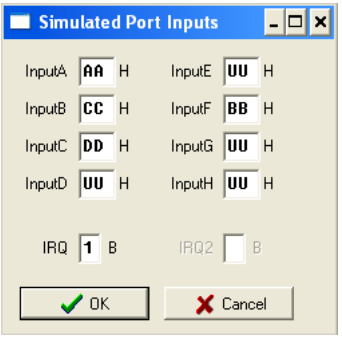
Show I/O port input values.

NOTE The IRQ pin state can be directly manipulated with IRQ command. For example, "IRQ 1" will simulate a high state on the IRQ pin; likewise, "IRQ 0" will simulate a logic-low state on the IRQ pin.

Keyboard Interrupt Module

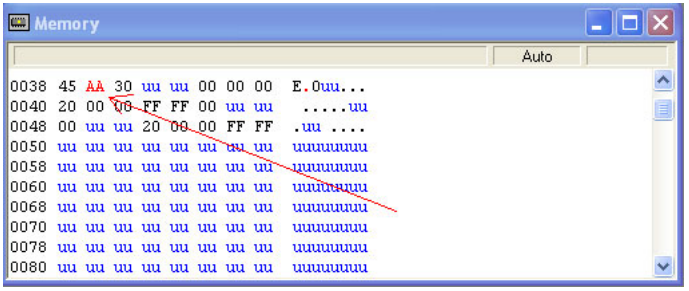
In Full Chip Simulation, this module will simulate all functionality of the Keyboard Interrupt (KBI) module including the edge-only, edge and level interrupt, and flag polling modes of operation. The Full Chip Simulation (FCS) mode uses simulated port inputs to trigger the KBI event from the proper I/O port pin. To define an input state of the specific port one should write the INPUT<x> <n> command in the Command line window. The <x> represents the corresponding I/O port, while <n> stands for the input value that should be written to this port. At the same time, the INPUTS command can be used to bring up the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports. Please see the documentation for **INPUT<x> Command** and INPUTS Command for more information about the various forms of this command.

Figure 20.22 Simulated Port Inputs Dialog Box



Using the Simulated Port Inputs Dialog, the input value to any I/O port can be reconfigured. In order to trigger the event, the inputs to the port should be manipulated in the appropriate manner, depending on whether the KBI is configured for edge-only or edge and level. Once the KBI event takes place, the KEYF Flag bit, which is a part of the Keyboard Status and Control register, can be observed in the Memory Window.

Figure 20.23 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

The user is able to poll the KBI Interrupt Pending flag if the Polling Mode is simulated. In Interrupt Mode, the simulator will branch to an appropriate interrupt subroutine as long as the KBI interrupt vector is properly configured. For more information on KBI configuration, please refer to the Freescale user manual corresponding to the microprocessor that you are using.

Keyboard Interrupt Commands

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax:

>INPUT<x> <n>

Where:

<x> is the letter representing corresponding port

<n> Eight-bit simulated value for port <x>

Example:

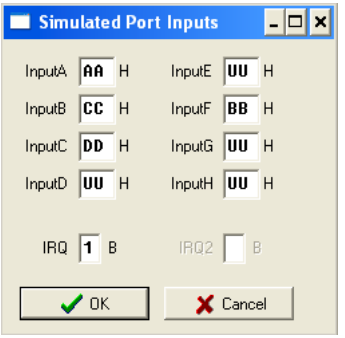
>INPUTA AA

Simulate the input AA on port A.

INPUTS Command

In Full Chip Simulation and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 20.24 on page 522. The use may then use this box to specify the input states of port pins and IRQ.

Figure 20.24 Simulated Port Inputs Dialog Box



When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax:

>INPUTS

Example:

>INPUTS

Show I/O port input values.

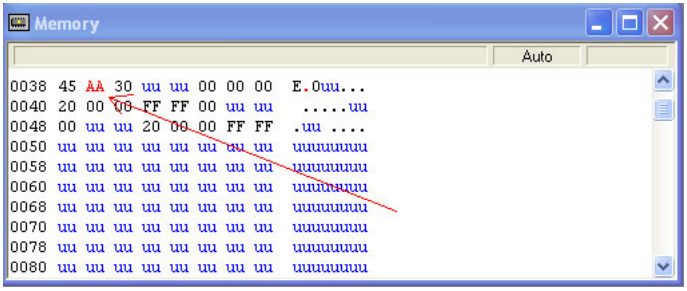
Modulo Timer Interrupt Module

In Full Chip Simulation, this module will simulate all functionality of the Modulo Timer Interrupt (MTIM) Module, including:

- programmable MTIM clock input
- free running or modulo up count operation
- flag polling
- interrupt enabled mode of operation

Once the MTIM Status and Control register properly configures the operation of the module, the MTIM Counter will start incrementing. If modulo up count operation is enabled, the MTIM overflow flag can be observed in the MTIM Status and Control register in the Memory Window.

Figure 20.25 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

If the MTIM interrupt is enabled, the Full Chip Simulation will jump to an appropriate subroutine as long as the MTIM interrupt vector is properly defined.

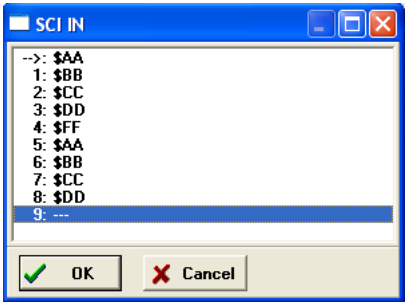
Serial Communications Interface Module

In Full Chip Simulation, this module will simulate all functionality of the Serial Peripheral Interface (SPI) module including:

- Flag polling
- Interrupt enabled mode
- 8- or 9-bit length data codes
- Odd and even parity modes
- Transmission and reception of external data

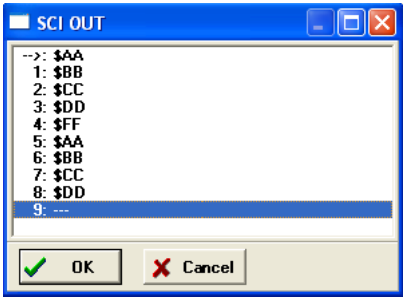
Full Chip Simulation (FCS) mode uses the buffered input/output structure to simulate SCI inputs. The user can queue up to 256 data values into the input buffer. The output buffer of the SCI module can also hold 256 output values. To queue the SCI Input Data, one should use the SCDI <n> command in the command prompt. If <n> (the data parameter) is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is provided, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the SCI. The data from the SCI input buffer is written to the SCI data register once the SCI module has been turned on and is properly configured for receiving data from an external serial device. Once the simulation of the data transmission is over, the arrow moves to the next value in the SCI IN Buffer.

Figure 20.26 SCI IN Buffer Display



SCI Data Output Buffer simulation allows the user to gain access to the past 256 SCI data values transmitted out of the module. To bring up the SCI OUT buffer dialog, the SCDO command should be used.

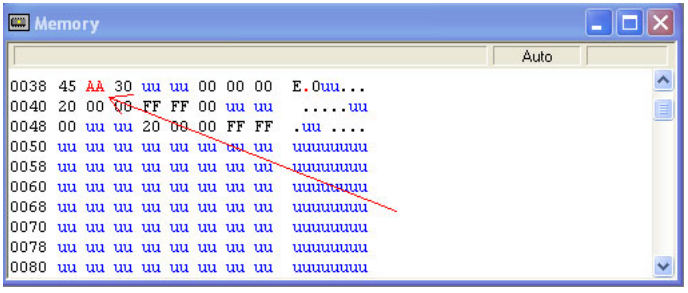
Figure 20.27 SCI OUT Buffer Display



At any point, the SCCLR command can flush the input and output SCI buffers.

After the SCI simulated input is received, the first queued value is passed from the data buffer into the SCI data register. It can be observed in the memory window by displaying the memory location corresponding to the SCI data register.

Figure 20.28 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

The user can also observe different SCI flags in the Memory window. If the module is run in Flag Polling mode, the user should poll the flag corresponding to the expected SCI event. If the SCI interrupts are enabled, the FCS will jump to an appropriate subroutine as long as the SCI interrupt vectors are properly defined.

For more information on how to configure SCI module for desired operation, refer to the Freescale user manual corresponding to the microprocessor that you are using.

SCI Commands

SCCLR Command

The SCCLR command can be used to flush the input and output buffers for SCI simulation. This will reset the buffers and clear out all values. Notice that if the SCI is in the process of shifting a value, this command will not prevent the SCI from finishing the

transfer. See SCDI command and SCDO command for accessing the input and output buffers of the SCI interface.

Syntax:

>SCCLR

Example

>SCCLR

Clear input and output buffer for SCI simulation

SCDI Command

The SCDI command allows the user to input data into the SCI. If a data parameter is given, the value is placed into the next slot in the SCI input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the SCI. The maximum number of input values is 256 bytes.

Syntax:

>SCDI [<n>]

Where:

><n> The value to be entered into the next location in the input buffer

Example:

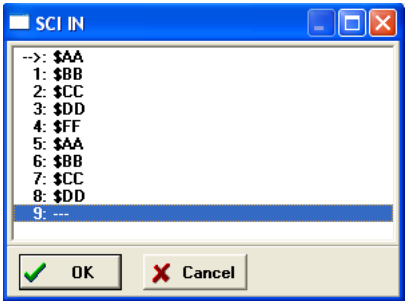
>SCDI $55

Set the next input value to the SCI to $55

>SCDI

Pull up the data window with all the input values.

Figure 20.29 SCI IN buffer display



SCDO Command

The SCDO command displays the output buffer from the SCI. A window is opened that shows all the data that the SCI has shifted out. An arrow is used to point to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

Syntax:

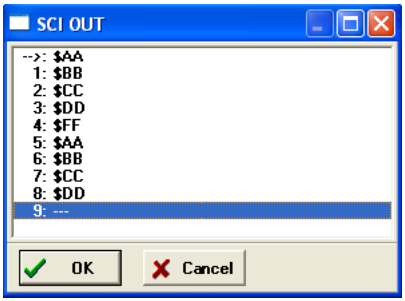
>SCDO

Example:

>SCDO

View data from the output buffer for the SCI simulation.

Figure 20.30 SCI OUT Buffer Display



Serial Peripheral Interface Module

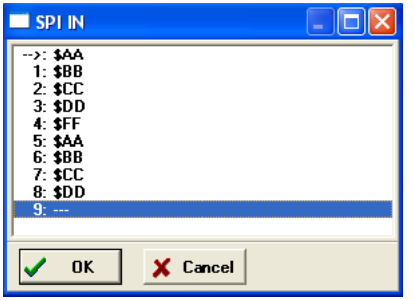
In Full Chip Simulation, this module will simulate all functionality of the Serial Peripheral Interface (SPI) module including:

- flag polling
- interrupt enabled mode
- master and slave modes
- slave input clock
- transmission and reception of external data

The Full Chip Simulation (FCS) mode uses the buffered input/output structure to simulate SPI inputs. The user can queue up to 256 data values into the input buffer. The output buffer of the SPI module can also hold 256 output values. To queue the SPI Input Data, one should use the SPDI <n> command at the command prompt. If <n> (the data parameter) is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is provided, a window is displayed with the input buffer values. Input values

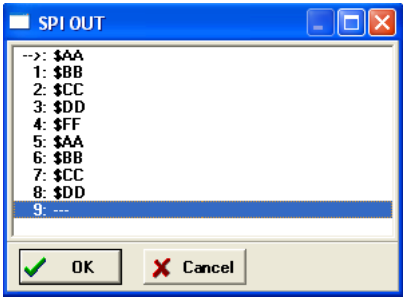
can be entered while the window is open. An arrow points to the value that will be used next as input to the SPI. The data from the SPI input buffer is written to the SPI data register once the SPI module has been turned on and is properly configured for receiving data from an external serial device. Once the simulation of the data transmission is over, the arrow moves to the next value in the SPI IN Buffer.

Figure 20.31 SPI IN Buffer Display



SPI data output buffer simulation allows the user to gain access to the past 256 SPI data values transmitted out of the module. To bring up the SPI OUT buffer dialog, the SPDO command should be used.

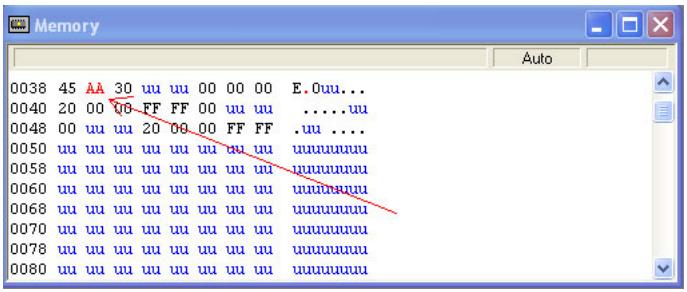
Figure 20.32 SPI OUT Buffer Display



At any point, the SPCLR command can flush the input as well as output SPI buffers.

After the SPI simulated input is received, the first queued value is passed from the data buffer into the SPI data register. It can be observed in the Memory Window by displaying the memory location corresponding to the SPI data register.

Figure 20.33 Memory Component Window


Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

The user can also observe different SPI flags, in the Memory window. If the module is run in the Flag Polling mode, the user should poll the flag corresponding to the expected SPI event. If the SPI interrupts are enabled, the FCS will jump to an appropriate subroutine as long as the SPI channel interrupt vectors are properly defined.

To simulate the frequency of the SPI slave input clock, the SPFREQ <n> command is used. If the SPI is configured for slave mode, this command allows the user to enter the number of cycles <n> in the period of the input clock. If the SPFREQ command is not used, then clocking is set by the SPI control register.

For more information on how to configure the SPI module for desired operation, refer to the Freescale user manual corresponding to the microprocessor that you are using.

SPI Commands

SPCLR Command

The SPCLR command can be used to flush the input and output buffers for SPI simulation. This will reset the buffers and clear out all values. Notice that if the SPI is currently shifting a value, this command will not prevent the SPI from finishing the transfer. See SPDI command and SPDO command for accessing the input and output buffers of the SPI interface.

Syntax:

>SPCLR

Example:

>SPCLR

Clear input and output buffer for SPI simulation

SPDI Command

The SPDI command allows the user to input data into the SPI. If a data parameter is given, the value is placed into the next slot in the SPI input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the value that will be used next as input to the SPI. The maximum number of input values is 256 bytes.

Syntax:

>SPDI [<n>]

Where:

<n> The value to be entered into the next location in the input buffer

Example:

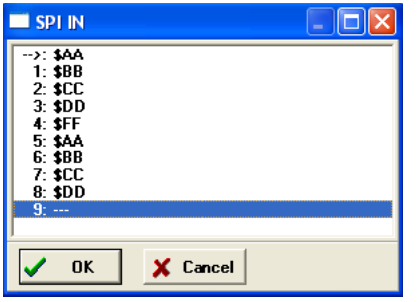
>SPDI $55

Set the next input value to the SPI to $55

>SPDI

Pull up the data window with all the input values.

Figure 20.34 SPI IN Buffer Display



SPDO Command

The SPDO command displays the output buffer from the SPI. A window is opened that shows all the data that the SPI has shifted out. An arrow is used to point to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

Syntax:

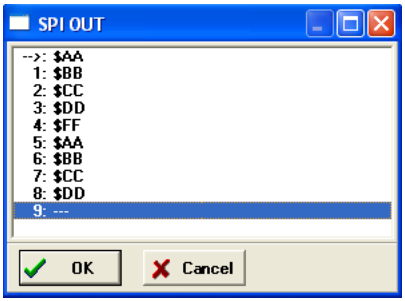
>SPDO

Example:

>SPDO

View data from the output buffer for the SPI simulation.

Figure 20.35 SPI OUT Buffer Display



SPFREQ Command

The SPFREQ command lets the user set the frequency of the SPI slave input clock. If the SPI is configured for the slave mode, this command allows the user to enter the number of cycles <n> per one input clock period. If no value is given, a popup window will appear and the user will be prompted for a value. If this command is not used, then the clocking is assumed to be set by the SPI control register.

Syntax:

>SPFREQ [<n>]

Where:

<n> The number of cycles for the period of the input clock.

Example:

>SPFREQ 8

Set the period of the input slave clock to 8 cycles (total shift = 8*8 cycles per bit = 64 cycles)

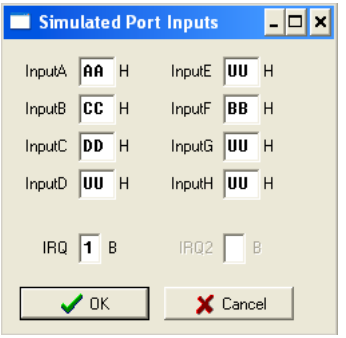
Timer Interface Module

In Full Chip Simulation, this module will simulate all functionality of the Timer Interface module, including:

- Input capture/output compare
- Pulse width modulation
- Internal or external clock input
- Free running or modulo up count operation
- Flag polling
- Interrupt enabled mode of operation.

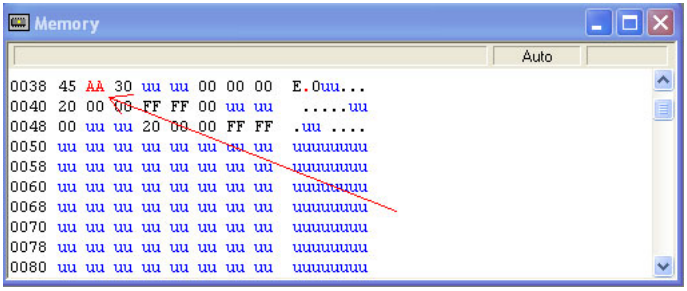
The Full Chip Simulation (FCS) mode uses the simulated port inputs to trigger the input capture on a given timer channel. To define an input state of the specific port one should use the INPUT<x> <n> command in the Command line window. The <x> represents the corresponding I/O port, while <n> stands for the input value that should be written to this port. At the same time, the INPUTS command can be used to display the Simulated Port Inputs for all general IO ports. It displays the current simulated values to all applicable input ports. Please see the documentation for INPUT<x> Command and INPUTS Command for more information about the various forms of this command.

Figure 20.36 Simulated Port Inputs Dialog Box



Using the Simulated Port Inputs dialog box, the input value to any I/O port can be reconfigured. Depending on whether the input capture is set for rising/falling edge, to trigger the event, the inputs to the port should first be set high/low and then be inverted to an opposite value. Once the Input Capture event takes place, the CHxF can be observed in the Channel Status and Control register in the Memory window.

Figure 20.37 Memory Component Window

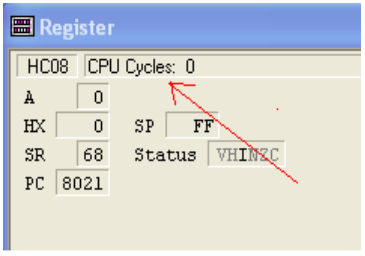

Memory
Auto
0038 45 AA 30 uu uu 00 00 00 E.0uu...
0040 20 00 00 FF FF 00 uu uu .....uu
0048 00 uu uu 20 00 00 FF FF .uu ....
0050 uu uu uu uu uu uu uu uu uuuuuuuu
0058 uu uu uu uu uu uu uu uu uuuuuuuu
0060 uu uu uu uu uu uu uu uu uuuuuuuu
0068 uu uu uu uu uu uu uu uu uuuuuuuu
0070 uu uu uu uu uu uu uu uu uuuuuuuu
0078 uu uu uu uu uu uu uu uu uuuuuuuu
0080 uu uu uu uu uu uu uu uu uuuuuuuu

If the Timer module is configured for an Output Compare event, once the event takes place the same CHxF Flag can be observed via the Memory window. If the timer channel interrupt is enabled, the FCS will jump to an appropriate subroutine as long as the Timer channel interrupt vector is properly defined. To observe the Timer Overflow Flag (TOF) flag going up as a result of the corresponding CPU event, situate your Memory window on the memory location of the Timer Status and Control register.

To observe the Pulse Width Modulation (PWM) operation, properly configure the Timer to operate in the Modulo up count mode, choose the toggle-on-overflow or clear/set output on compare events to create a desired duty cycle wave. Once a PWM event takes place, pin toggle/clear/set behavior corresponding to the Timer configuration can be observed in the Memory window displaying the IO port associated with a given timer channel.

To observe the accuracy of the Timer module operation, the user can observe the number of CPU cycles that it takes for the event to occur. The cycle counter is only incremented as the user steps through the code. To determine the exact amount of cycles over which the event occurs, one can either observe the cycle display in the Register window or use the built in simulation commands. To display the current number of cycles in the Command window, the CYCLES command should be used. To change the number of cycles in the cycle counter, use CYCLES <n>, where <n> is the new cycle value. If the number of cycles that an event should take has already been pre-calculated, one can use CYCLE 00 to reset the number of cycles and GOTOCYCLE <n> to run through the code until the place where expected event should take place.

Figure 20.38 Register Window With Cycles Display



Timer Module Commands

CYCLES Command

The CYCLES command changes the value of the cycles counter. The cycles counter counts the number of the processor cycles that have passed during execution. The Cycles Window shows the cycle counter. The cycle count can be useful for timing procedures.

Syntax:

>CYCLES <n>

Where:

<n> Integer value for the cycles counter

Examples:

>CYCLES 0

Reset cycles counter

>CYCLES 1000

Set cycle counter to 1000.

GOTOCYCLE Command

The GOTOCYCLE command executes the program in the simulator beginning at the address in the program counter (PC). Execution continues until the cycle counter is equal to or greater than the specified value, until a key or the Stop button on the toolbar is pressed, until it reaches a break point, or until an error occurs.

Syntax:

GOTOCYCLE <n>

Where:

<n> Cycle-counter value at which the execution stops

Example:

>GOTOCYCLE 100

Execute the program until the cycle counter equal 100.

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax:

INPUT<x> <n>

Where:

<x> is the letter representing corresponding port

<n> Eight-bit simulated value for port <x>

Example:

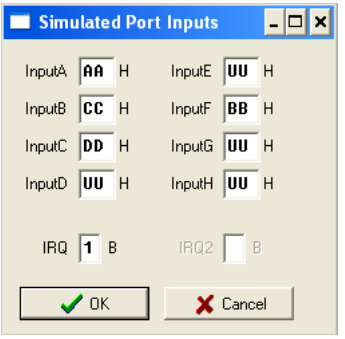
>INPUTA AA

Simulate the input AA on port A.

INPUTS Command

In Full Chip Simulation and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 20.39 on page 536. The use may then use this box to specify the input states of port pins and IRQ.

Figure 20.39 Simulated Port Inputs Dialog



When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax:

>INPUTS

Example:

>INPUTS

Show I/O port input values.

21

HCS08 P&E Multilink/ Cyclone PRO Connection

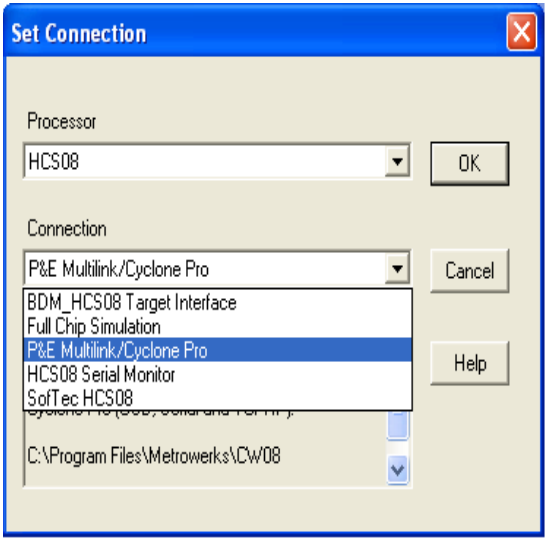
The HCS08 P&E Multilink/Cyclone Pro Connection setting permits a connection to Multilink/Cyclone PRO devices. HCS08 P&E Multilink/Cyclone Pro connection mode allows the user to debug code, as the firmware is fully resident in the FLASH of the microprocessor. The operation of all modules fully reflects the actual operation of the on-board resources.

Connection Procedure

To select the P&E Multilink/Cyclone Pro as your debugger connection:

1. Choose the P&E Multilink/Cyclone Pro option from the set connection dialog box as shown in Figure 21.1 on page 537.
2. Click the OK button.

Figure 21.1 Set Connections Dialog Box



3. Choose the P&E device that you are using from the Interface drop-down menu and click on Refresh. See Figure 21.2 on page 538 and Figure 21.3 on page 539.

Figure 21.2 HCS08 Connection Assistant Interface Selection

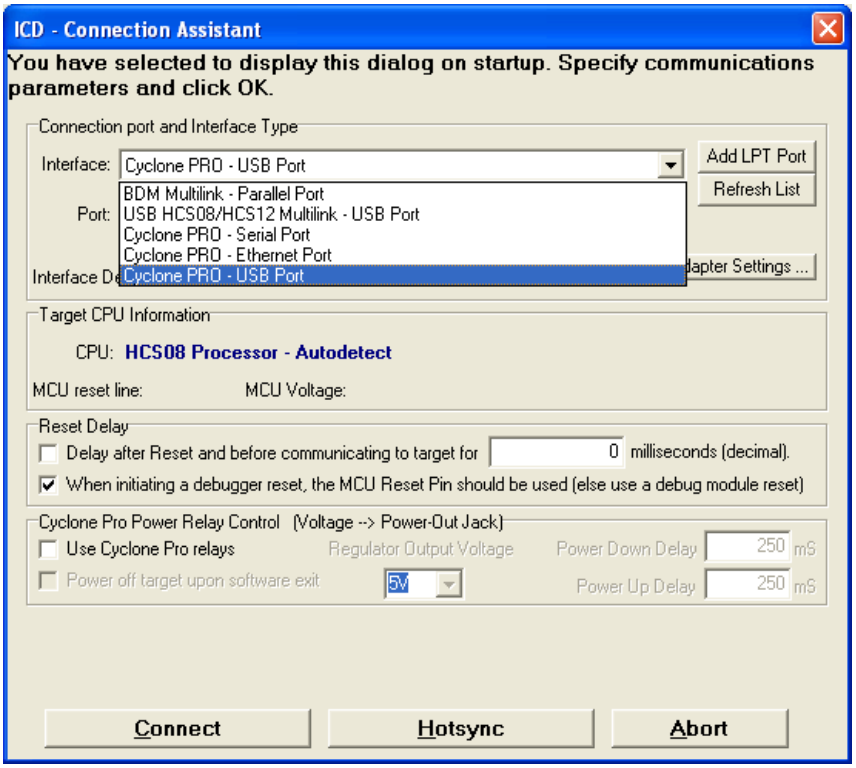
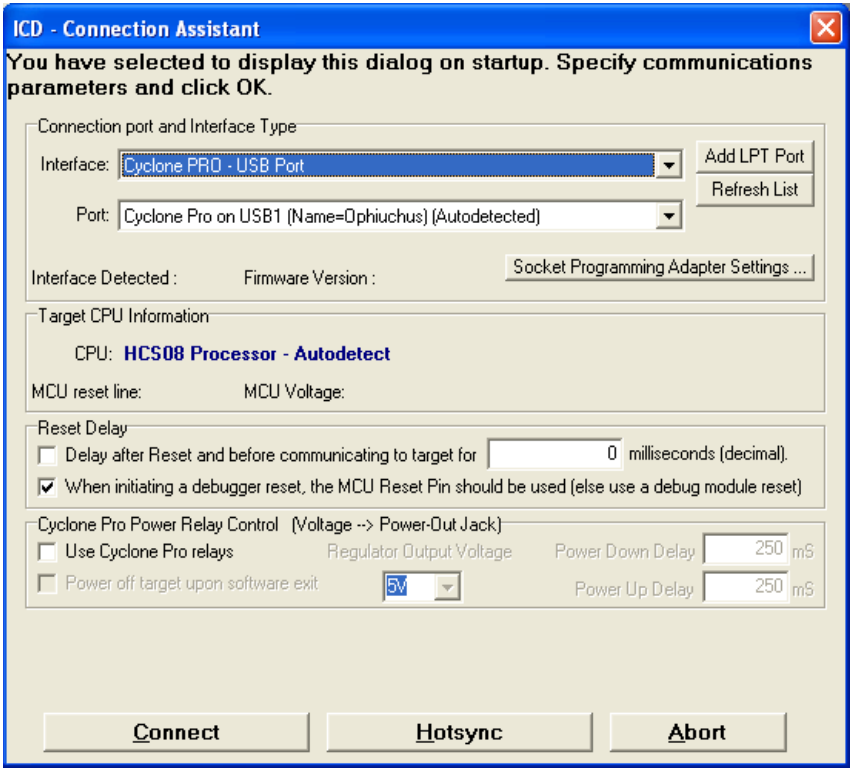


Figure 21.3 HCS08 Connection Assistant Interface Selected



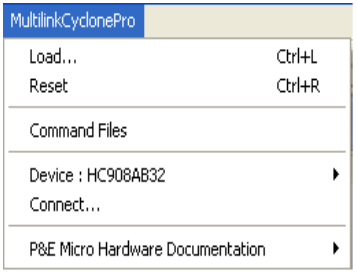
Hotsync Button

The “Hotsync” button in the Connection Assistant (see Figure 21.3 on page 539) allows the user to connect to an already running target.

MultilinkCyclonePro Menu Description

When you have selectd P&E Multilink/Cyclone Pro as your connection, the Connection menu's name is changed and other options are added.

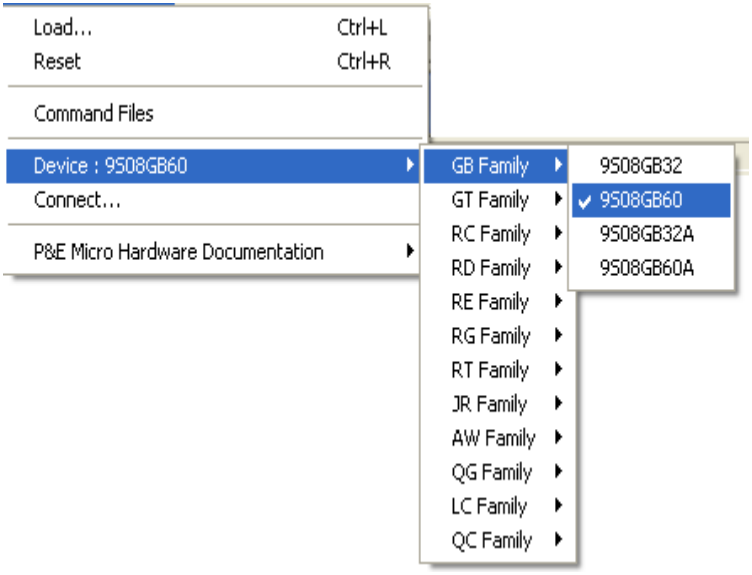
Figure 21.4 Connection (MultilinkCyclonePro) Menu



Device Option

The "Device" option in the MultilinkCyclonePro menu allows the user to select the particular Freescale processor that they wish to use. When choosing Device from the MultilinkCyclonePro menu, extended menus will open which allow you to select the family (e.g. GB Family), and device type (e.g. 9S0GB60) of the MCU that you are using.

Figure 21.5 HCS08 Device Extended Menu



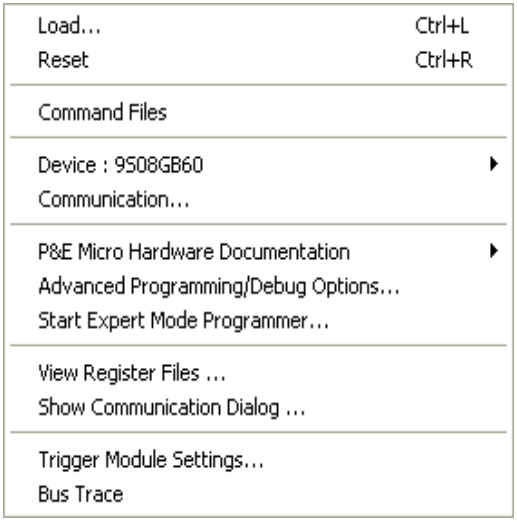
Connect Option

The “Connect” option initiates an attempt to communicate with the device chosen under the device section of the menu.

Active Mode Menu Options

When the microprocessor is connected, more Connection menu options become available to the user.

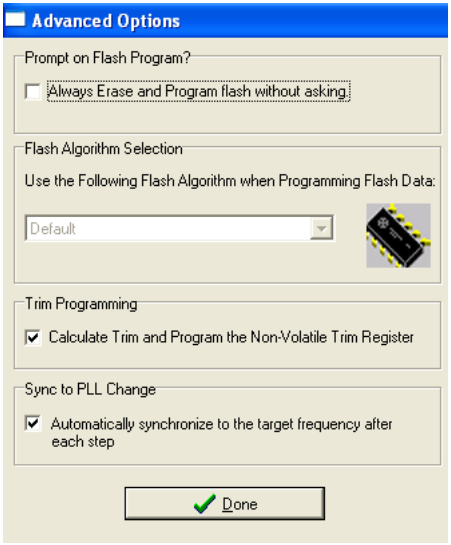
Figure 21.6 Additional Connection Menu Options



Advanced Programming/Debug Options Option

The Advanced Programming/Debug Options menu option takes you to the Advanced Options dialog box, where you can configure the software settings for the FLASH programming procedure.

Figure 21.7 Advanced Options Dialog Box


Advanced Options
Prompt on Flash Program?
Always Erase and Program flash without asking
Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
Default
Trim Programming
Calculate Trim and Program the Non-Volatile Trim Register
Sync to PLL Change
Automatically synchronize to the target frequency after each step
Done

Prompt on Flash Program Checkbox

Checking “Always Erase and Program flash without asking” in this dialog box lets the software transparently program the microprocessor.

Trim Programming Checkbox

The “Trim Programming” checkbox enables automatic calculation and programming of the trim value in a designated Non-Volatile memory location.

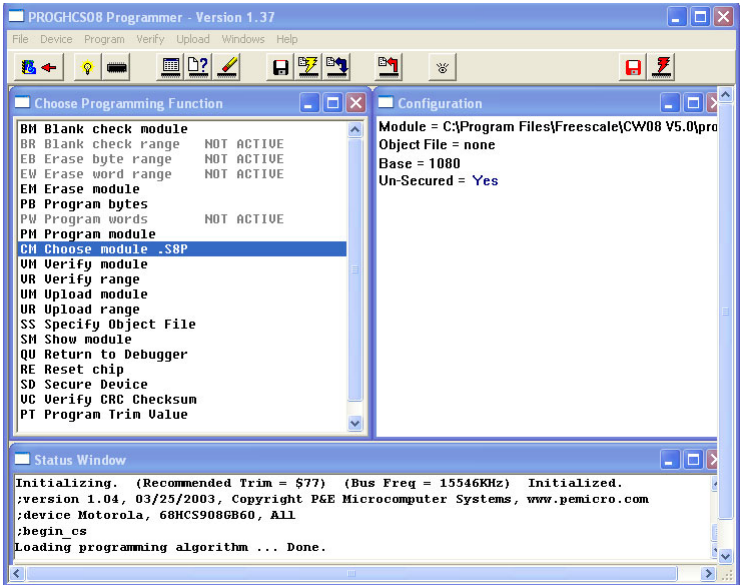
Sync to PLL Change Checkbox

“Sync to PLL Change” is required for the software/hardware connection to synchronize with the microprocessor during the flash erasing/programming procedure.

Start Expert Mode Programmer Option

The Start Expert Mode Programmer option of the Connection Menu grants to the user access to P&E’s graphical flash programming utility, PROGHCS08. PROGHCS08 lets an advanced user control the step-by-step execution of the flash erase/programming procedure. See Figure 21.8 on page 544. More information on how to use the PROGHCS08 can be found on the P&E Microcomputer Systems website at www.pemicro.com.

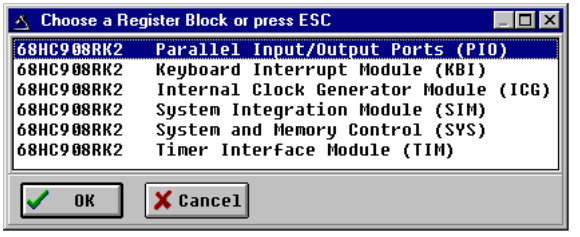
Figure 21.8 PROGHCS08 Programmer Window


PROGHCS08 Programmer - Version 1.37
File Device Program Verify Upload Windows Help
Choose Programming Function
BM Blank check module
BR Blank check range NOT ACTIVE
EB Erase byte range NOT ACTIVE
EW Erase word range NOT ACTIVE
EM Erase module
PB Program bytes
PW Program words NOT ACTIVE
PM Program module
CM Choose module .S8P
VM Verify module
VR Verify range
UM Upload module
UR Upload range
SS Specify Object File
SM Show module
QU Return to Debugger
RE Reset chip
SD Secure Device
VC Verify CRC Checksum
PT Program Trim Value
Configuration
Module = C:\Program Files\Freescale\CW08 V5.0\pro
Object File = none
Base = 1080
Un-Secured = Yes
Status Window
Initializing. (Recommended Trim = $77) (Bus Freq = 15546KHz) Initialized.
;version 1.04, 03/25/2003, Copyright P&E Microcomputer Systems, www.pemicro.com
;device Motorola, 68HCS908GB60, All
;begin_cs
Loading programming algorithm ... Done.

View Register Files Option

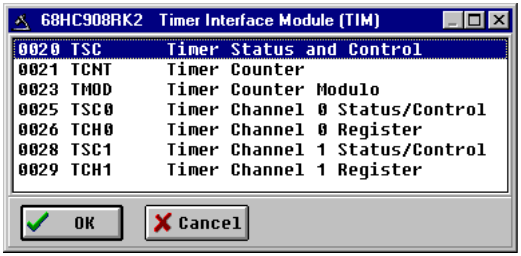
The “View Register Files...” Connection menu selection also gives the user the option of running the register file viewer/editor. If register files are available for the device that you have chosen, the “Choose a Register Block” window (see Figure 21.9 on page 544) will be opened. You may also open it by entering the R command in the Command Window command line.

Figure 21.9 Choose A Register Block Window


Choose a Register Block or press ESC
68HC908RK2 Parallel Input/Output Ports (PIO)
68HC908RK2 Keyboard Interrupt Module (KBI)
68HC908RK2 Internal Clock Generator Module (ICG)
68HC908RK2 System Integration Module (SIM)
68HC908RK2 System and Memory Control (SYS)
68HC908RK2 Timer Interface Module (TIM)
OK
Cancel

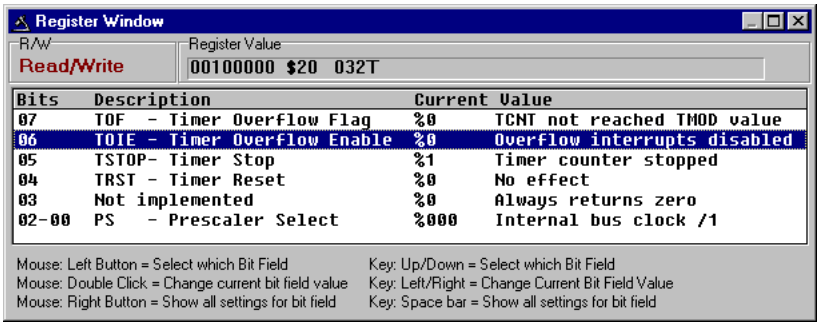
If register files have been installed on the host computer, selecting a block brings up the Register Block register listing (see Figure 21.10 on page 545), which shows a list of the associated registers, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.

Figure 21.10 Register Block Register Listing


68HC908RK2 Timer Interface Module (TIM)
0020 TSC Timer Status and Control
0021 TCNT Timer Counter
0023 TMOD Timer Counter Modulo
0025 TSC0 Timer Channel 0 Status/Control
0026 TCH0 Timer Channel 0 Register
0028 TSC1 Timer Channel 1 Status/Control
0029 TCH1 Timer Channel 1 Register
OK
Cancel

Selecting a file brings up the Register Window (see Figure 21.11 on page 545), which displays the values and significance for each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.

Figure 21.11 Register Window


Register Window
R/W
Read/Write
Register Value
00100000 $20 032T
Bits Description Current Value
07 TOF - Timer Overflow Flag %0 TCNT not reached TMOD value
06 TOIE - Timer Overflow Enable %0 Overflow interrupts disabled
05 TSTOP- Timer Stop %1 Timer counter stopped
04 TRST - Timer Reset %0 No effect
03 Not implemented %0 Always returns zero
02-00 PS - Prescaler Select %000 Internal bus clock /1
Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

Trigger Module Settings Option

For more information about the “Trigger Module Settings...” menu option, please see **Trigger Module Settings Window**.

Bus Trace Option

For more information about the “Bus Trace” menu option, please see Trace Modes.

22

Softec HCS08 Connection

This section guides you through the first steps toward debugging with CodeWarrior and the *SofTec HCS08* connection. It does not replace all the additional documentation provided in this manual, but gives you a good starting point.

SofTec HCS08 Technical Considerations

The 8/16 bits debugger (and then the CodeWarrior IDE) might be connected to HCS08 hardware using the SofTec HCS08.

When the debugger runs the **SofTec HCS08** connection, it can communicate and debug **HCS08** core based hardware connected throught the SofTec in-circuit debugger/ programmer units, i.e:

SofTec Microsystems HCS08 ISP Debuggers/Programmers (inDART Series) and Starter Kits (PK and newer Series).

Please refer to the "***inDART®-HCS08 In-Circuit Debugger/Programmer for Motorola HCS08 Family FLASH Devices User's Manual***" from SofTec for communication hardware requirements and SofTec product installation.

CodeWarrior and SofTec HCS08 Connection

There are two seperate paths that may be followed to take the first steps toward debugging with Codewarrior and the SofTec inDART-HCS08 connection. The differences between the two paths hinge on the starting point for the steps:

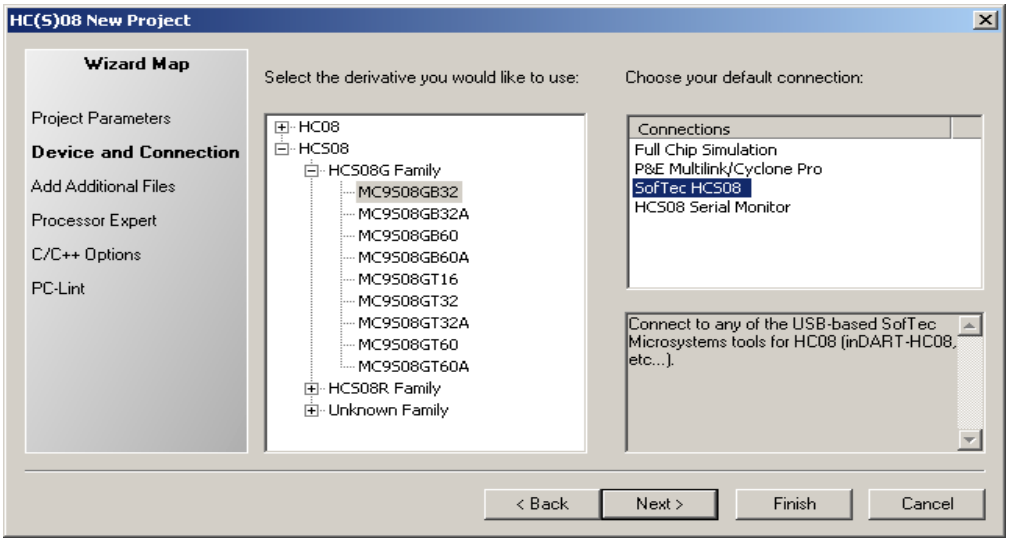
- Using the Stationary Wizard at the start of the project
- From within an existing project

First Steps Using the Stationery Wizard

To take the first steps toward debugging with CodeWarrior and theSofTec inDART-HCS08 using the stationery Wizard:

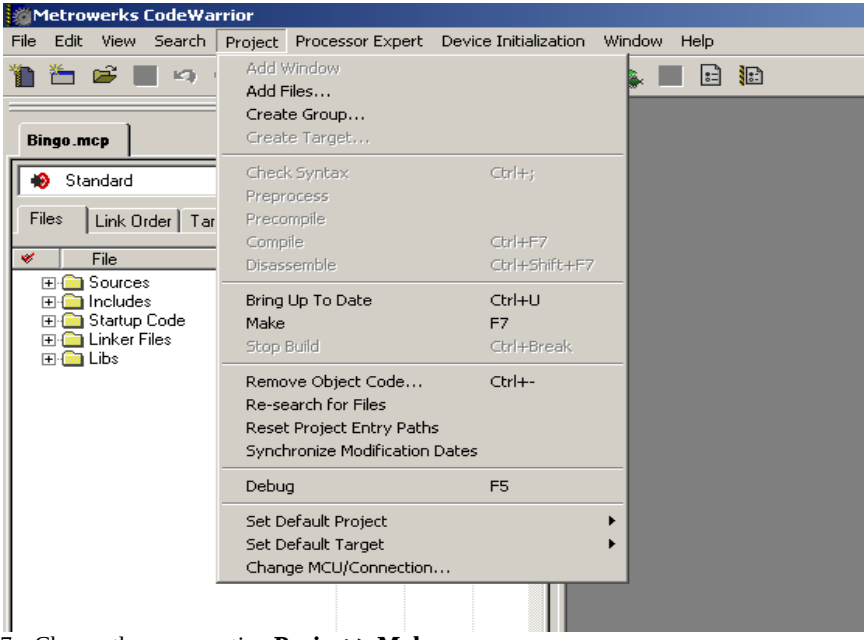
1. Run the *CodeWarrior IDE* with the shortcut created in the program group.
2. In the *HC(S)08 New Project Wizard*, follow the path to create a new project, naming the project.
3. Click the Next button to open the New Project window.
4. In the HC(S)08 New Project window, choose HCS08 Family chip you are working with from the list in the Derivative list box in the left of the window.

Figure 22.1 Wizard Connection Selection



5. From the Default Connection list box, choose the connection **SofTec HCS08** to create a new project from this stationery.
6. Click the Finish button - IDE opens as shown in Figure 21.2, "IDE Main Window - Project Menu" on page 549.

Figure 22.2 IDE Main Window - Project Menu



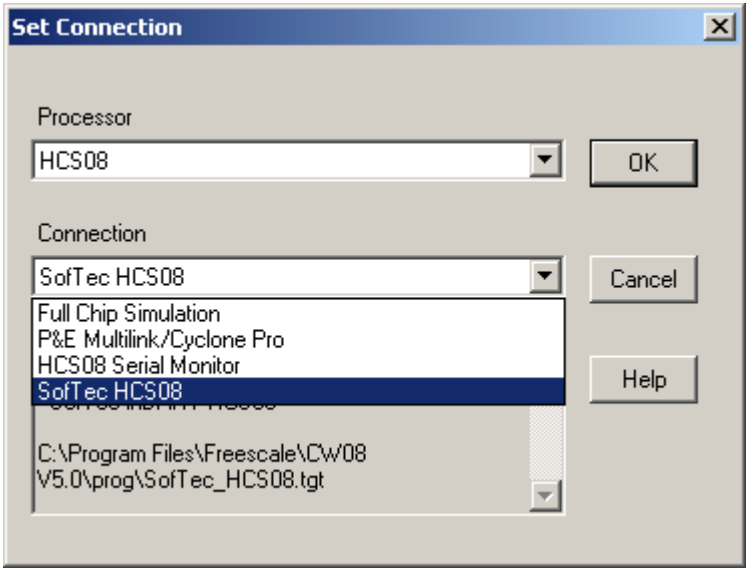
7. Choose the menu option **Project > Make.**
8. Choose the menu option **Project > Debug** to start the debugger.
9. Start debugging.

First Steps From Within an Existing Project

To take the first steps toward debugging with CodeWarrior and setting the SofTec HCS08 connection from within an existing debugging project:

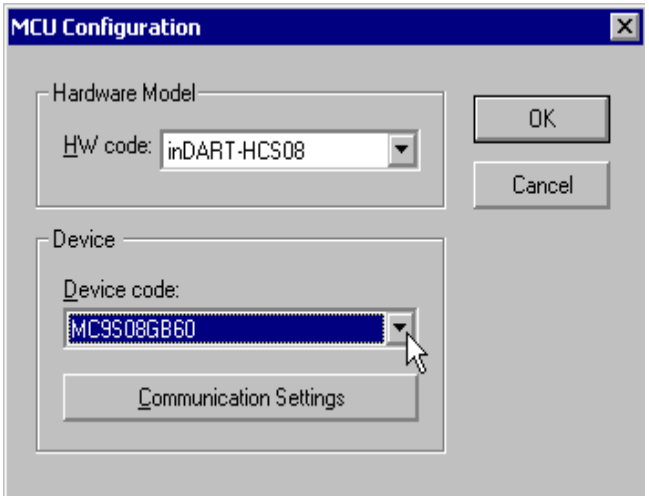
1. Run the *CodeWarrior IDE* with the shortcut created in the program group.
2. Open the project.
3. Choose the menu **Project > Debug** to start the debugger.
4. Choose in the debugger menu **Component > Set Connection..** to select another target interface in the Set Connection dialog box.
5. Select ***HCS08*** as Processor then ***SofTec HCS08*** as connection.

Figure 22.3 Set Connection Dialog Box - SofTec HCS08 Selection



6. Press the OK button - MCU Configuration dialog box opens.
7. In the MCU Configuration dialog box, choose the correct target processor.

Figure 22.4 MCU Configuration Dialog Box

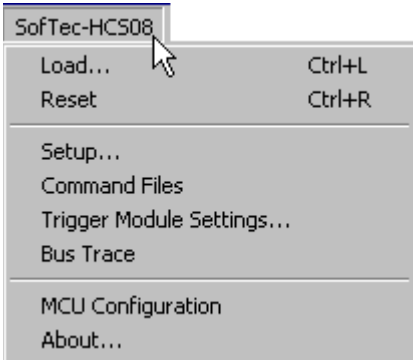


8. Press the OK button to start debugging.

SofTec HCS08 Menu Options

Once the ***SofTec HCS08*** connection is set, the connection menu entry in the debugger main toolbar is "***SofTec-HCS08***".

Figure 22.5 SofTec-HCS08 Menu Options



MCU Configuration Option

Select entry **SofTec-HCS08 > MCU Configuration** to display the MCU Configuration Dialog Box.

About Option

Select the **SofTec-HCS08 > About...** *option* to display the About Dialog Box.

Trigger Module Settings

Select the **SofTec-HCS08 > Trigger Module Settings...** *option* to force immediate mass erasure of the target processor flash. Refer to the "HCS08 On-chip DBG Module" on page 567 section to get find all related information.

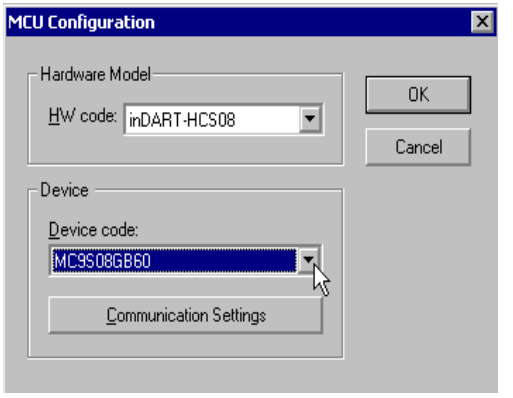
Bus Trace

Select the **SofTec-HCS08 > Bus Trace** *option* to open the Trace component window within the debugger main window.Refer to the "HCS08 On-chip DBG Module" on page 567 section to get find all related information.

MCU Configuration Dialog Box

The Hardware Model drop down list can be expanded to select another type of BDC debug interface than the SofTec inDART-HCS08. The *HW Code* drop down list can be expanded to select another HCS08 derivative. Note that at this document release time, only the SofTec inDART-HCS08 is available.

Figure 22.6 MCU Configuration Dialog Box



Pressing the *Communication Settings* button in this window opens the Communication Settings Dialog Box.

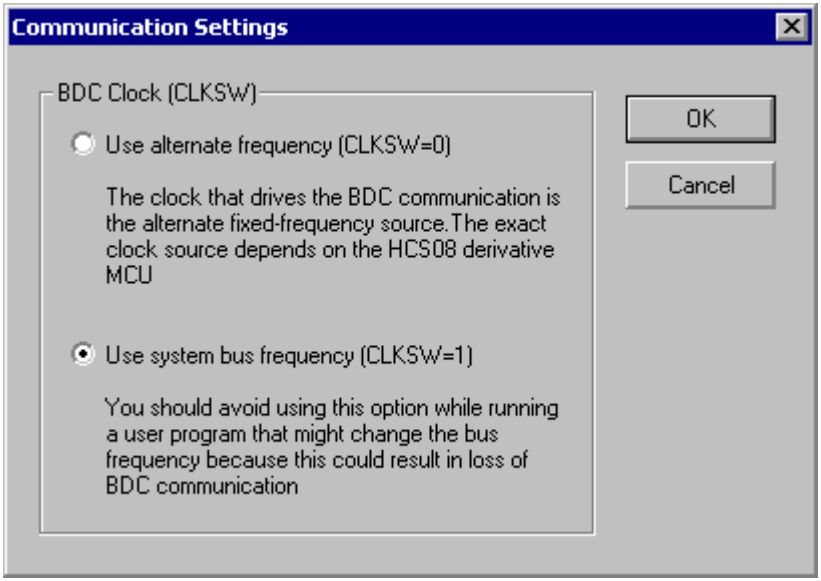
Communication Settings Dialog Box

The *BDC Clock (CLKSW)* group is intended to setup the best BDC synchronization between the *SofTec inDART-HCS08* interface and the target processor.

When "***Use system bus frequency***" is selected, the BDC communication clock source is the microcontroller's bus frequency; when "***Use alternate frequency***" is selected, the BDC communication clock source is a constant clock source, which can vary depending on the specific HCS08 derivative. In the case of the MC9S08GB60, for example, this constant clock source is an 8 MHz internal clock. Other derivatives may use the external crystal frequency.

Refer to the ***"inDART®-HCS08 In-Circuit Debugger/Programmer for Motorola HCS08 Family FLASH Devices User's Manual"*** from SofTec for further details.

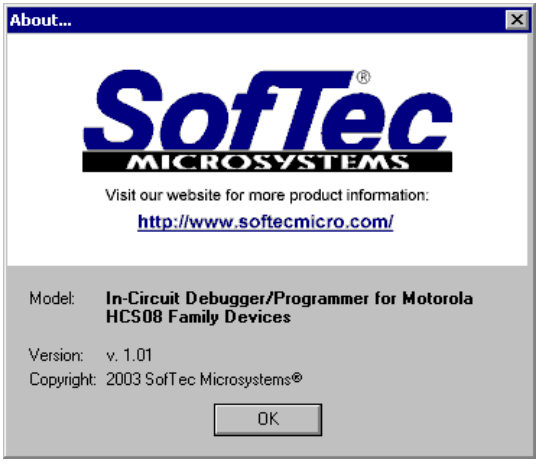
Figure 22.7 Communication Settings Dialog box


Communication Settings
BDC Clock (CLKSW)
Use alternate frequency (CLKSW=0)
The clock that drives the BDC communication is the alternate fixed-frequency source.The exact clock source depends on the HCS08 derivative MCU
Use system bus frequency (CLKSW=1)
You should avoid using this option while running a user program that might change the bus frequency because this could result in loss of BDC communication
OK
Cancel

About Dialog Box

This dialog box belongs to the SofTec GDI DLL and provides information about the ***SofTec_BDC08.dll*** release and version.

Figure 22.8 About Dialog Box



23

HCS08 Serial Monitor Connection

This section guides you through the first steps toward debugging with CodeWarrior and the *HCS08 Serial Monitor* connection. It does not replace all the additional documentation provided in this manual, but gives you a good start.

Serial Monitor Technical Considerations

The 8/16 bit debugger (and then the CodeWarrior IDE) might be connected to HCS08 hardware using the HCS08 Seial Monitor connection. This connection supports communication specifications described in the ***Serial Monitor for MC9S08GB/GT Application Note AN2140/D*** from Motorola (now Freescale)

When the debugger runs the HCS08 Seial Monitor connection, it can communicate and debug hardware running the HCS08 Serial Monitor in full compliance with the Motorola (now Freescale) ***Serial Monitor for MC9S08GB/GT Application Note AN2140/D*** specifications. Please refer to this Application Note for communication hardware requirements.

CodeWarrior and Serial Monitor Connection

There are two seperate paths that may be followed to take the first steps toward debugging with Codewarrior and the HCS08 Serial Monitor connection. The differences between the two paths hinge on the starting point for the steps:

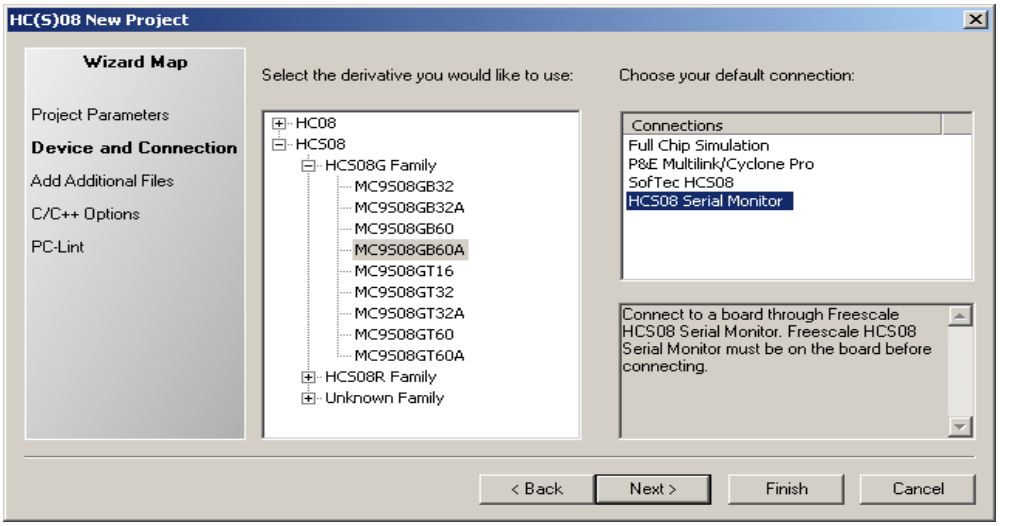
- Using the Stationary Wizard at the start of the project
- From within an existing project

First Steps Using the Stationery Wizard

To take the first steps toward debugging with CodeWarrior and the HCS08 Serial Monitor connection:

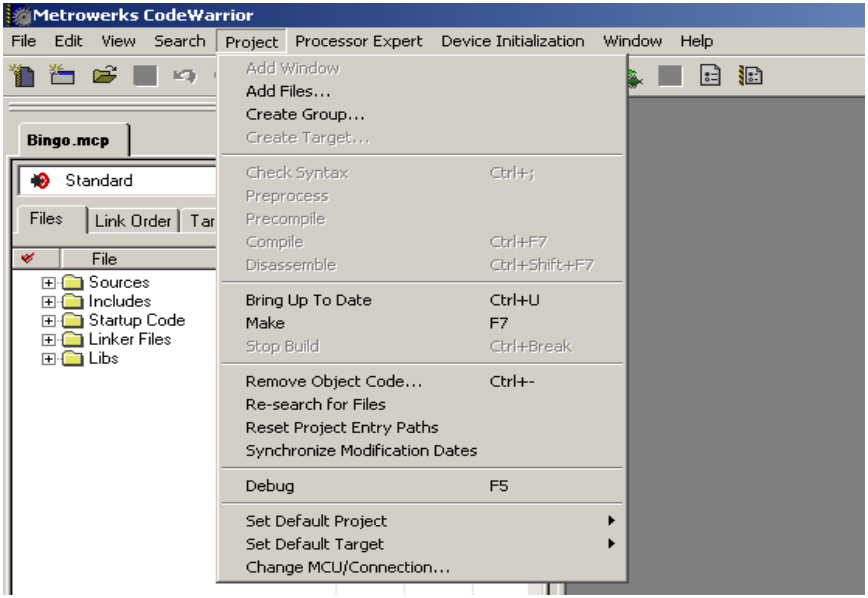
1. Run the *CodeWarrior IDE* with the shortcut created in the program group.
2. In the *HC(S)08 New Project Wizard*, follow the path to create a new project, naming the project.
3. Click the Next button to open the New Project window.
4. In the HC(S)08 New Project window, choose HCS08 Family chip you are working with from the list in the Derivative list box in the left of the window.

Figure 23.1 HC(S)08 New Project Wizard Window



5. From the Default Connection list box, choose the connection **HCS08 Serial Monitor** to create a new project from this stationery.
6. Click the Finish button - IDE opens as shown in Figure 22.2, "IDE Main Window - Project Menu" on page 557.

Figure 23.2 IDE Main Window - Project Menu



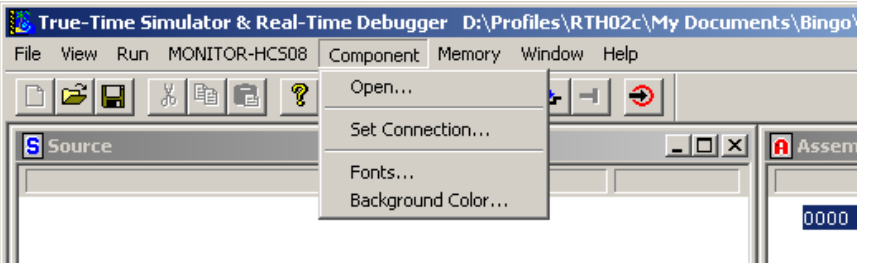
7. Choose the menu option **Project > Make.**
8. Choose the menu option **Project > Debug** to start the debugger.
9. Start debugging.

First Steps From Within an Existing Project

To take the first steps toward debugging with CodeWarrior and setting the HCS08 Serial Monitor connection from within an existing debugging project:

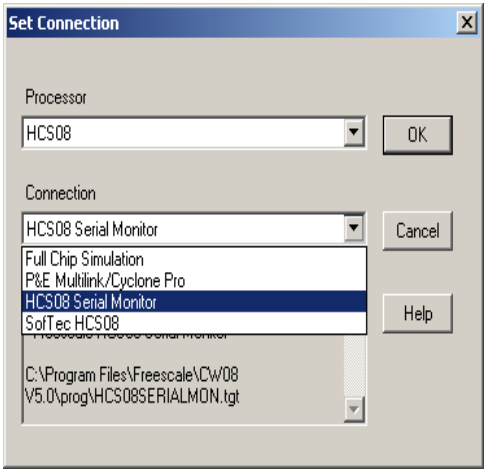
1. Run the *CodeWarrior IDE* with the shortcut created in the program group.
2. Open the existing project.
3. Choose the menu **Project > Debug** to start the debugger - debugger main window opens.

Figure 23.3 Debugger Main Window - Component Menu



4. In the debugger main window, from the Component menu, choose **Component > Set Connection...** to select another connection.

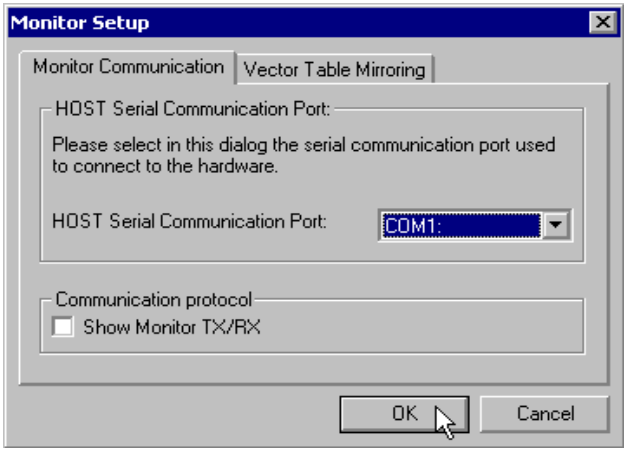
Figure 23.4 Set Connection Dialog Box - HCS08 Serial Monitor Selection



5. Select ***HCS08*** as Processor then ***HCS08 Serial Monitor*** as connection in the Set Connection dialog box and click the OK button.

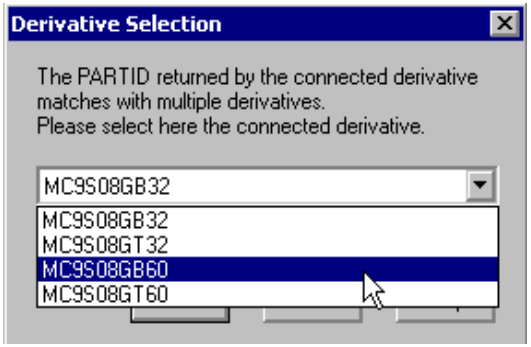
6. Now in the Monitor Setup window, Monitor Communitation tab, choose the correct Host serial communication port if necessary.

Figure 23.5 Monitor Setup Window - Monitor Communication Tab



7. Press the OK button. The HCS08 Serial Monitor connection reads the device silicon ID. This ID can match several derivatives.
8. Set the correct derivative matching with your hardware in the Derivative Selection dialog box.

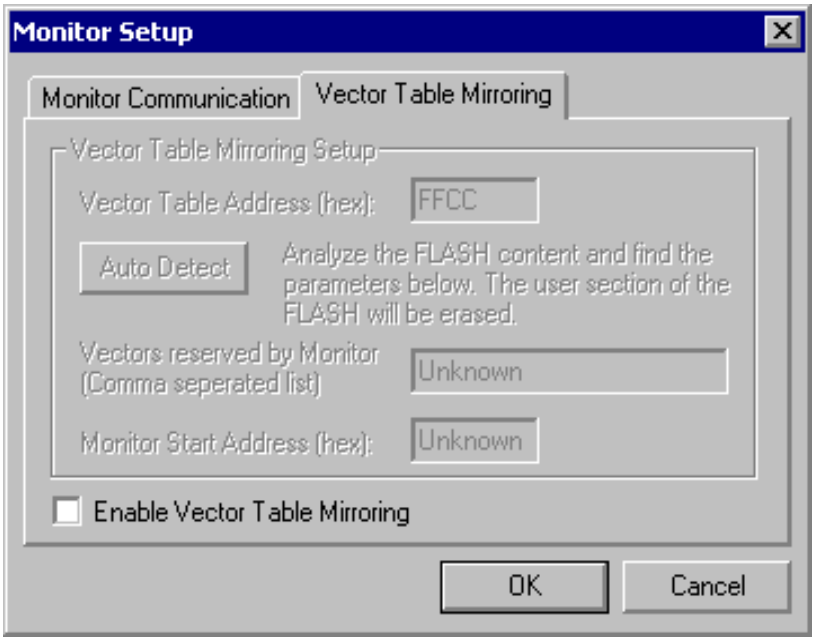
Figure 23.6 Derivative Selection Dialog Box



9. Press the OK button. The **Monitor Setup** window is opened again, to propose to use the “mirrored vector table” feature. See section ***Vector Redirection*** in the ***Serial Monitor for MC9S08GB/GT Application Note AN2140/D*** for all details. We recommand that you use the Vector Table Mirroring feature. Otherwise, vectors

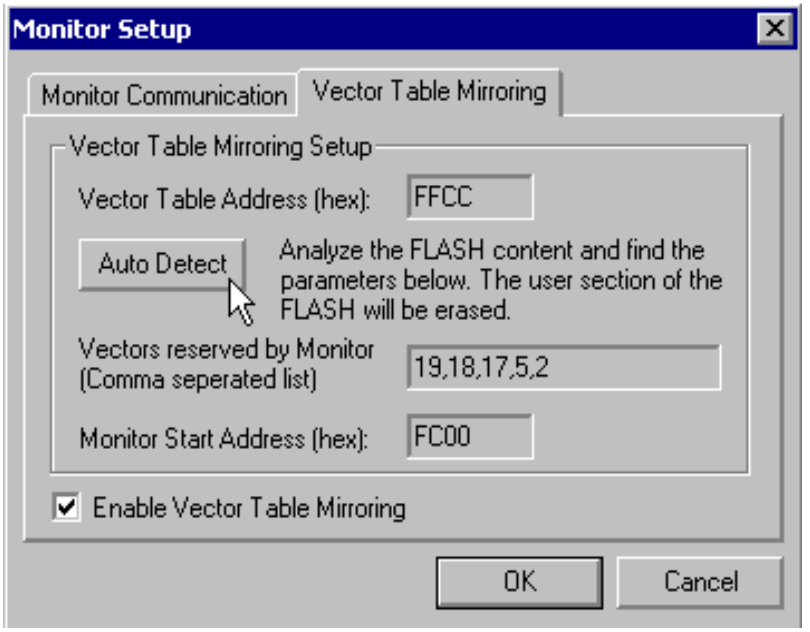
cannot be programmed as captured and protected from erasing or overwriting by the HCS08 Serial Monitor.

Figure 23.7 Monitor Setup Window - Vector Table Mirroring Tab



10. To enable this specific feature, check the “Enable Vector Table Mirroring” checkbox. ¨

Figure 23.8 Monitor Setup Window - Vector Table Mirroring Tab

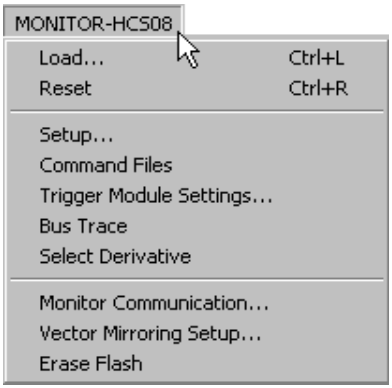


11. Press the “Auto Detect” button to make the debugger search for the vector table address and vectors reserved by the HCS08 Serial Monitor.
12. Once the autodetection succeeded, press the OK button to start debugging.

MONITOR-HCS08 Menu Options

Once the ***HCS08 Serial Monitor*** connection is set, the "*MONITOR-HCS08*" menu entry is is set in the Debugger menu, as shown in Figure 22.9 MONITOR-HCS08 Menu Entries.

Figure 23.9 MONITOR-HCS08 Menu Entries



Monitor Communication...

Select the **MONITOR-HCS08> Monitor Communications...** option to display the Monitor Setup Window - Monitor Communication Tab.

Vector Mirroring Setup...

Select the **MONITOR-HCS08> Vector Mirroring Setup...** *option* to display the Monitor Setup Window - Vector Table Mirroring Tab.

Erase Flash

Select the **MONITOR-HCS08> Erase Flash** *option* to force immediate mass erasure of the target processor flash.

Trigger Module Settings...

Select the **MONITOR-HCS08> Trigger Module Settings...** *option* to open the ***Trigger Module Settings*** dialog. Refer to the "***Debugger HCS08 Onchip DBG Module User Interface***" manual for all related information.

Bus Trace

Selectthe **MONITOR-HCS08> Bus Trace** toption o open the Trace component window within the debugger main window. Refer to the "***Debugger HCS08 Onchip DBG Module User Interface***" manual for all related information.

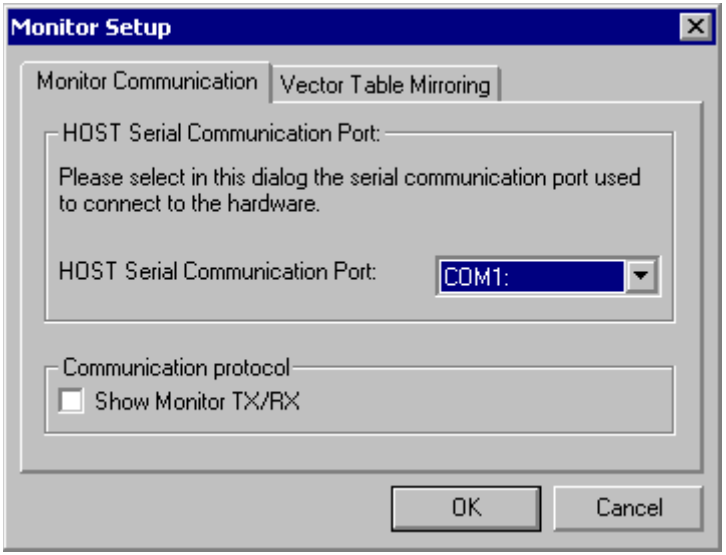
Select Derivative

Select the **MONITOR-HCS08> Select Derivative** option to open the Derivative Selection Dialog Box.

Monitor Setup Window

The Monitor Setup window has two tabs, as shown in Figure 22.10 Monitor Setup Window - Monitor Communication Tab and Figure 22.11 Monitor Setup Window - Vector Table Mirroring Tab.

Figure 23.10 Monitor Setup Window - Monitor Communication Tab

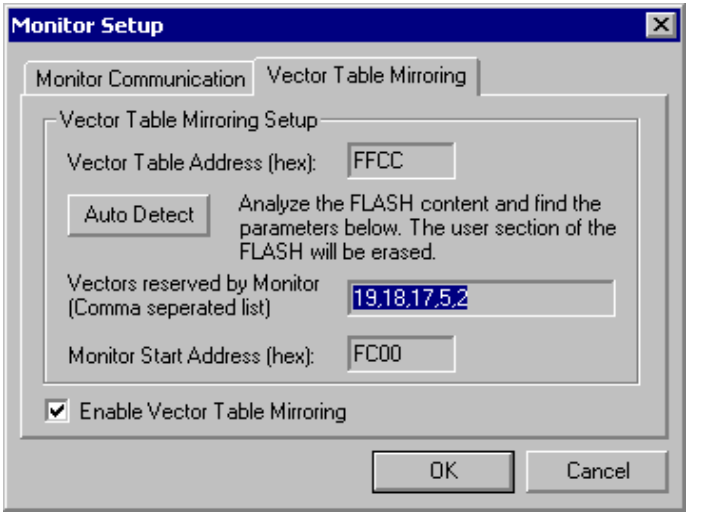


Monitor Communication Tab

Using the Monitor Communitaction tab, it is possible to set or modify the current serial communication port when opening the "*HOST Serial Communication Port*" list box's drop down list.

Checking the "*Show Monitor TX/RX*" checkbox, reports in the debugger Command Line window all low level communication frames between the host computer and the HCS08 Serial Monitor.

Figure 23.11 Monitor Setup Window - Vector Table Mirroring Tab



Vector Table Mirroring Tab

Using the Vector Table Mirroring tab, it is possible to set the "Vector Table Mirroring" feature. See the ***Vector Redirection*** section of Freescale ***Serial Monitor for MC9S08GB/GT Application Note AN2140/D*** for all details.

The HCS08 Monitor start address is given in the ***Monitor Start Address*** edit box.

The real vector table address is given in the ***Vector Table Address*** edit box.

The list of vectors reserved by the HCS08 Serial Monitor is given in the ***Vectors reserved by Monitor*** edit box.

NOTE In the Vectors reserved by Monitor list box above, the number "1" matches the **RESET** vector, "2" is the **SWI** vector, "5" is the **ICG** vector, etc.

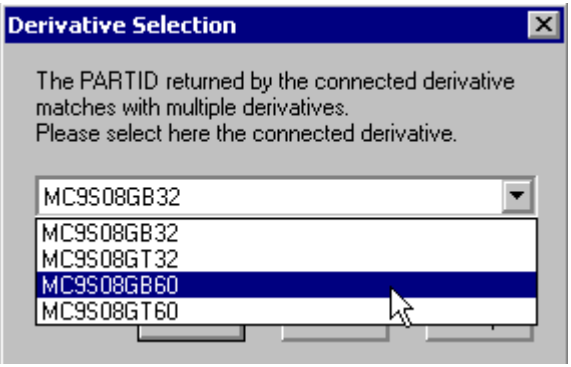
Vector table mirroring allows you to access chip vectors transparently. Indeed, the HCS08 Serial Monitor also uses some vectors, and the vector area is protected from erasing and overwriting. We recommend that you use this feature. Otherwise, user application vectors cannot be programmed as captured and are not protected from erasing/overwriting by the HCS08 Serial Monitor.

To enable this feature, check the “Enable Vector Table Mirroring” checkbox, then press the “Auto Detect” button to make the debugger search for the vector table address and vectors reserved by the HCS08 Serial Monitor. Once autodetection has succeeded, you can press the OK button to save and quit this window.

Derivative Selection Dialog Box

Within this dialog box, it is possible to select a specific derivative according to the *System Device Identification Register* (SDIDH, SDIDL) (also sometimes called PARTID) returned by the silicon device.

Figure 23.12 Derivative Selection Dialog Box



As several silicon devices might return the same value, a selection list is available to select the debugged derivative according to text reference written on the top of the silicon.

24

HCS08 On-chip DBG Module

The HCS08 derivatives featuring an on-chip DBG module require a debugger graphical user interface to setup this module and take full advantage of this enhanced debugging feature. This manual describes the debugger DBG module user interface.

Within several HCS08 debugger connections (e.g. P&E (PEDebug), HCS08 Serial Monitor and inDART-HCS08, a complete graphical user interface is provided, through a trigger setup dialog box combined with context sensitive popup menus (mouse right-click) in Source, Assembly, Data and Memory component windows to set the on-chip DBG module and triggers.

This DBG module support is automatically enabled or disabled, according to user selected derivative (if the device is user configurable) or automatically through device Part Id.

Reference Document

The HCS08 on-chip DBG module described in this manual is also described in:

-" ***HCS08RMv1/D, Rev. 1, 6/2003***" Motorola (now Freescale) document.

DBG Features

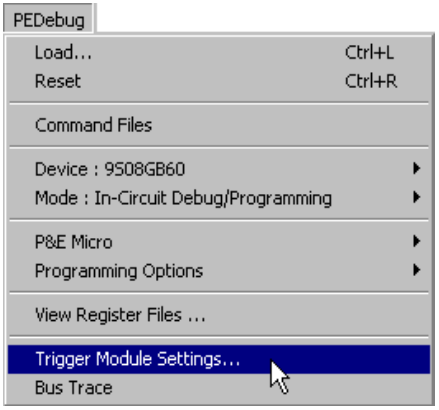
The debugger covers all features available within the on-chip DBG module:

- Regular hardware breakpoints and watchpoints,
- Predefined preset "Instruction" Triggers on page 587, "Memory Access" Triggers on page 585 or "Capture" Triggers on page 590, a wide set of complex hardware breakpoints (triggers on program code instructions) and watchpoints (triggers on device memory access) and data bus recording,
- Expert Triggers, as powerful as predefined preset triggers, "Do It Yourself" way,
- Code program flow rebuild from DBG data capture within the Trace window component (the Trace component should be opened to display the code program flow rebuild),
- Real time program code profiling and coverage within the Profiler and Coverage window components (the Profiler and/or the Coverage components should be opened to display code profiling and code coverage)

Specific Connection Menu Options

Specific DBG support menu options are added to the Connection menu as soon as the debugger target processor is acknowledged by the DBG module. Two additional popup menu entries are displayed: "***Trigger Module Settings***" and "***Bus Trace***" in the Connection menu. Shown below is an example with the P&E (PEDebug) connection.

Figure 24.1 Connection Menu - Added DBG Options



Choose "***Trigger Module Settings***" to open the Trigger Module Settings Window on page 580.

Choose "***Bus Trace***" to open the Trace Component Window on page 597.

Popup Menu Entries in Source, Data, Assembly and Memory Windows

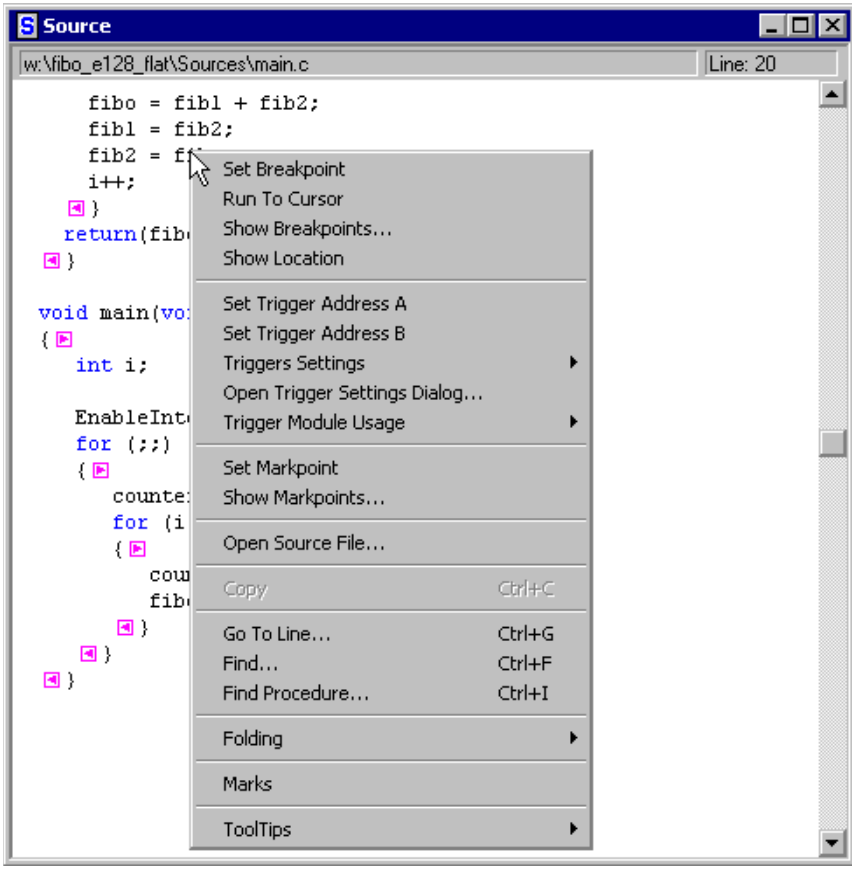
Specific DBG support menu options are added to the Connection menu as soon as the debugger target processor is acknowledged by the DBG module.

Source and Assembly Windows

Source and Assembly windows have menu entries to set/delete "Instruction" Triggers on page 587 A and/or B, a Trigger Settings on page 578 to set the DBG module Triggers Settings and the Trigger Module Usage on page 579 to set the DBG module functionality globally.

Setting a trigger, which can be assimilated as a complex breakpoint or watchpoint, is as simple as setting a breakpoint.

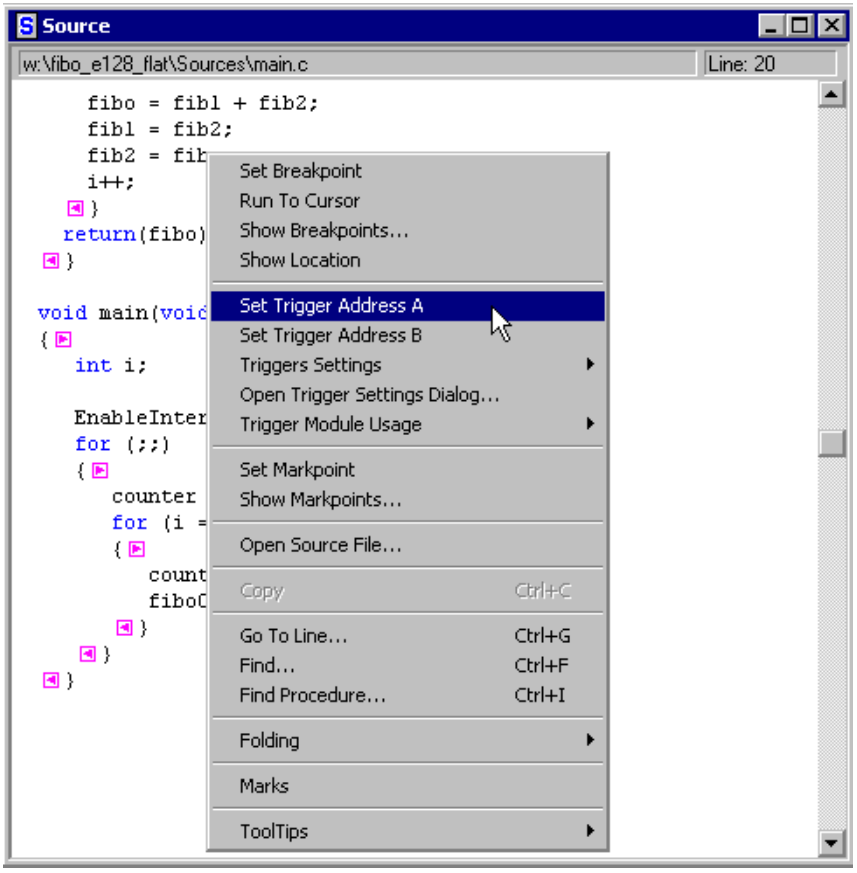
Figure 24.2 Source Popup Menu - Added Options



Instead of setting a breakpoint, a trigger can be set. Note that only 2 triggers can be set: ***Trigger A*** and ***Trigger B***. In a general way, the on-chip DBG module provides combinations of trigger A and trigger B conditions, and according to the number of triggers defined (one or two), different triggers DBG Module Mode Setup on page 581 can be chosen.

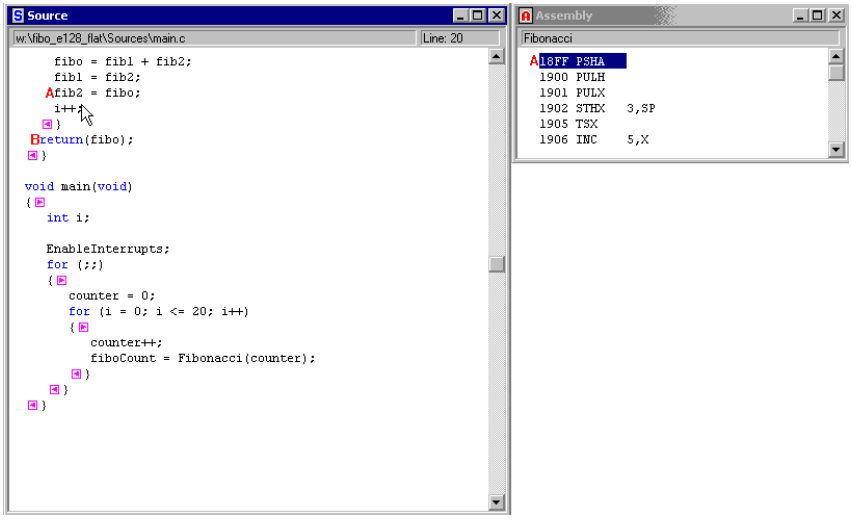
To set a trigger, choosing a ***Set TriggerAddress*** entry sets a trigger at the selected source location/address.

Figure 24.3 Set Trigger Address A Option



The trigger is displayed in the Source window and at the corresponding address in the Assembly window, just like a breakpoint icon. To be distinguishable from breakpoints, the trigger A is marked with a red "**A**" icon and trigger B with a red "**B**" icon.

Figure 24.4 Triggers Set in Source and Assembly Windows



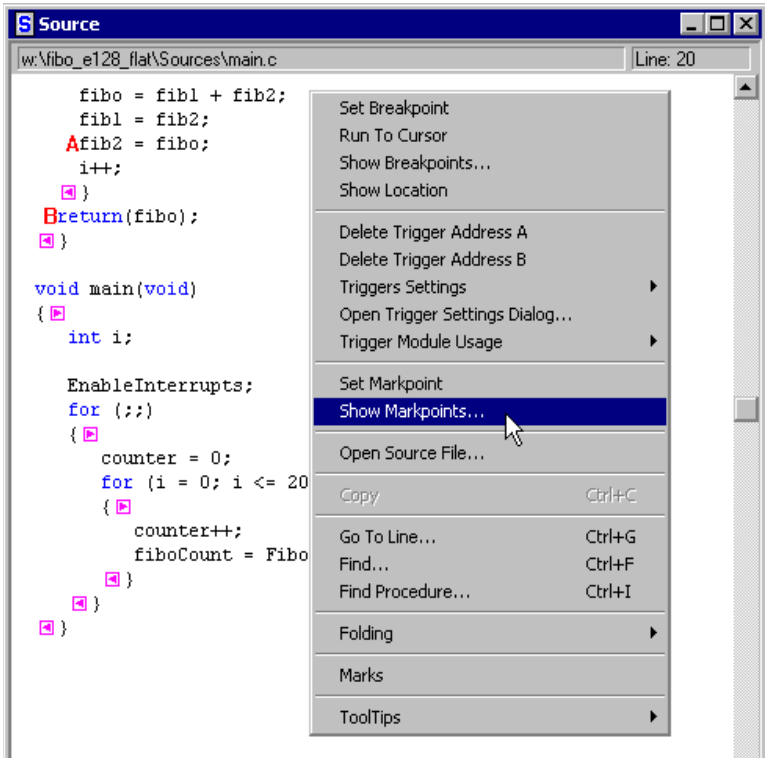
Once a trigger is set, it can be deleted by opening any context sensitive popup menu that contains the ***Delete Trigger Address*** options.

Trigger Storing as Markpoints

Triggers are stored in the debugger as special markpoints. Like breakpoints, markpoints can be viewed on choosing ***Show Markpoints...*** in the menu.

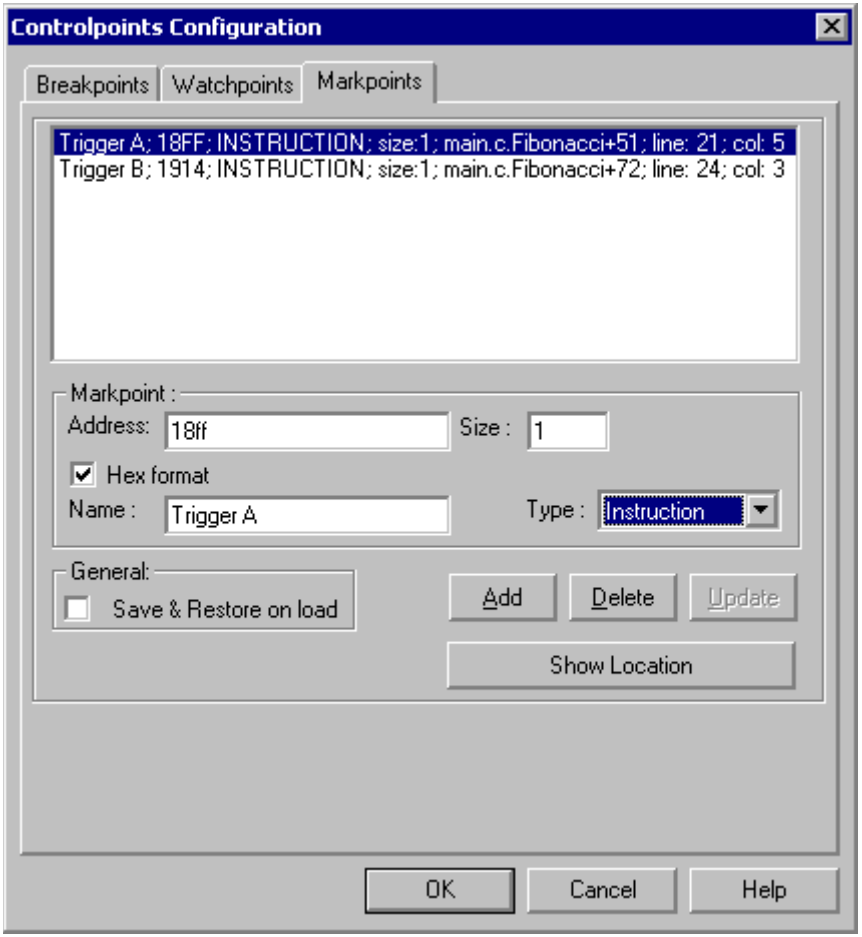
Triggers are stored as *"**Trigger A**"* and *"**Trigger B**"* markpoints. These markpoint names are therefore reserved by the debugger. The markpoint type "**INSTRUCTION**" is automatically selected when the trigger was set from the Source or the Assembly window.

Figure 24.5 Show Markpoints Option



Selecting the Show Markpoints option from the Source window causes the ***Controlpoints Configuration*** window to open with its ***Markpoints*** tab displayed.

Figure 24.6 Controlpoints Configuration Window - Markpoints Tab



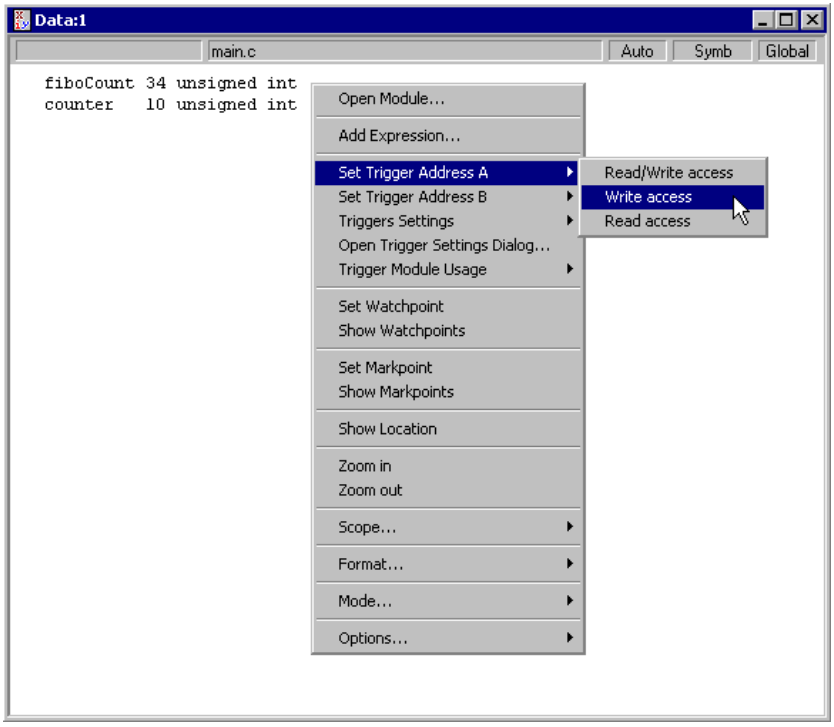
Editing triggers through the ***Markpoints*** tab in the ***Controlpoints Configuration*** window below is not user friendly. However, the "***Save and Restore on load***" option (also available with breakpoints and watchpoints) can be very useful to automatically save the application with the DBG module setup and trigger positions saved as they are set in the current application.

Data and Memory Windows

Data and Memory windows have popup menu options to set/delete "Memory Access" Triggers on page 585 A and/or B, a Trigger Settings on page 578 option to set the DBG module triggers settings and the Trigger Module Usage on page 579 option to set globally the DBG module functionality.

Setting a trigger,which can be assimilated as a complex breakpoint or watchpoint, is as simple as setting a watchpoint.

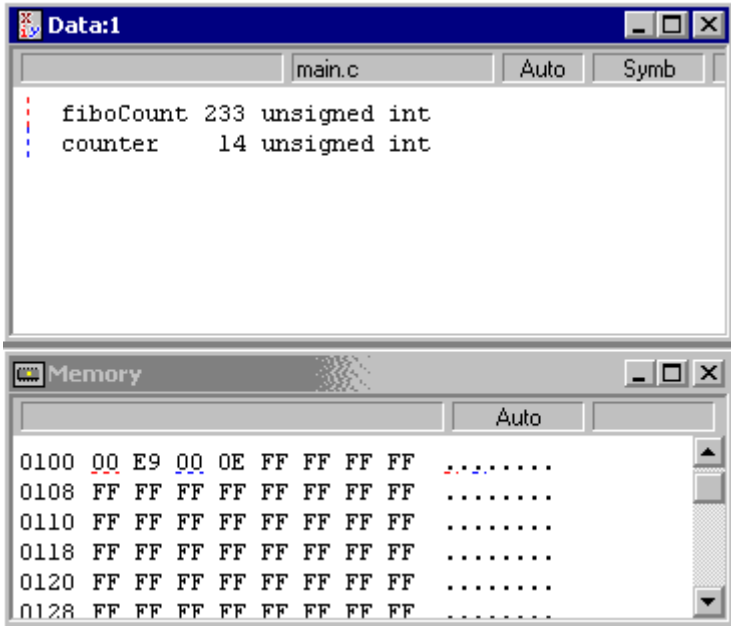
Figure 24.7 Data Window Popup Menu - Set Trigger A Option



In the Data window, instead of setting a watchpoint, a trigger can be set. Note that only 2 triggers can be set: Trigger A and trigger B. In a general way, the on-chip DBG module provides combinations of trigger A and trigger B conditions, and according to the number of triggers defined (one or two), a different trigger DBG Module Mode Setup on page 581 can be chosen.

To set a trigger, choosing a ***Set Trigger Address*** entry and the kind of access - **Read, Write, Read/Write** - sets a trigger at the selected place.

Figure 24.8 Triggers Set in Data and Memory Windows


Data:1
main.c
Auto
Symb
fiboCount 233 unsigned int
counter 14 unsigned int
Memory
Auto
0100 00 E9 00 0E FF FF FF FF ....
0108 FF FF FF FF FF FF FF FF ....
0110 FF FF FF FF FF FF FF FF ....
0118 FF FF FF FF FF FF FF FF ....
0120 FF FF FF FF FF FF FF FF ....
0128 FF FF FF FF FF FF FF FF ....

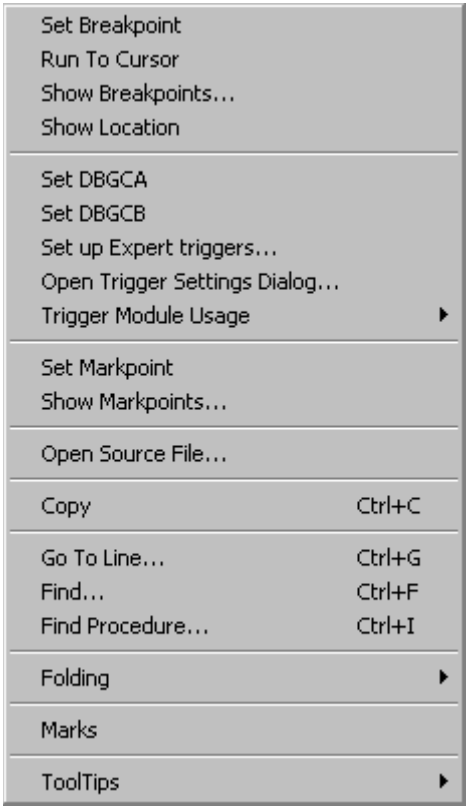
The trigger is displayed in the Data window and at the corresponding address in the Memory window like a watchpoint icon. To be distinguished from watchpoints, the trigger A is marked with a red dotted vertical line and trigger B with a blue dotted vertical line.

Expert Triggers

The Expert Mode on page 582 has a different set of triggers and trigger designs. Indeed, to completely separate the Expert mode from the Automatic mode, the debugger provides a second set of triggers for the Expert mode. Expert Triggers on page 575 are independant from the regular triggers described previously.

Context sensitive popup menu entries are slightly different, basically replacing the "***Set Trigger Address A***" entry by a "***Set DBGCA***" entry and the "***Set Trigger Address B***" entry by a "***Set DBGCB***" entry. The renaming is due to a more physical DBG registers approach in Expert mode and in the Expert Mode Tab on page 583.

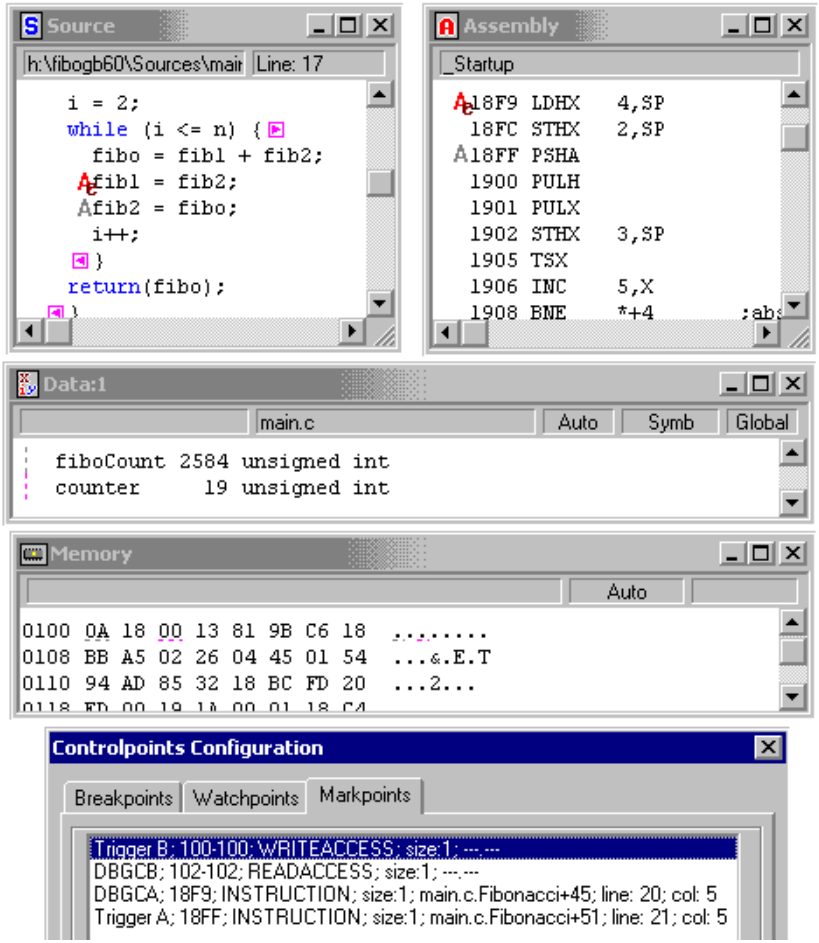
Figure 24.9 Source Window Popupu Menu - Expert Trigger Options



As shown in the next picture, Expert triggers are displayed in Source and Assembly windows with a small additional "**e**" character and different colors in the Memory component.

NOTE When the Expert mode is set, preset "Instruction" Triggers on page 587, "Memory Access" Triggers on page 585 or "Capture" Triggers on page 590 designs are grayed out. When the automatic mode is set or a predefined preset trigger is set, the Expert mode trigger designs are grayed out.

Figure 24.10 Expert Triggers in Source, Assembly, Memory and Data Windows



As shown above, expert triggers are stored in the Markpoints tab of the Controlpoints Configuration window as *"**DBGCA**" and "**DBGCB**"* markpoints. These markpoint names are therefore reserved by the debugger.

The markpoint type "**INSTRUCTION**" is automatically selected when the trigger was set from the Source or the Assembly window.

As for regular triggers, the markpoint types "**READACCESS**", "**WRITEACCESS**" or "**READWRITEACCESS**" are automatically selected when the trigger was set from the Data or the Memory window.

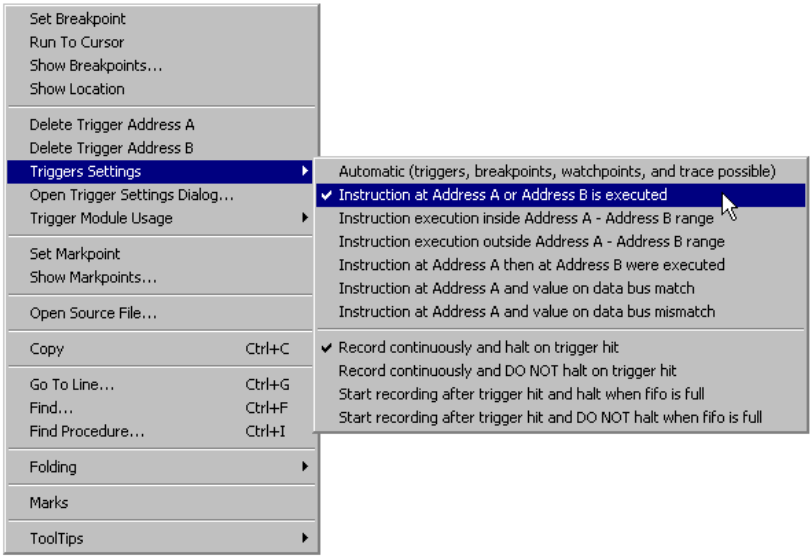
Just as with regular triggers, editing expert triggers through the ***Markpoints*** tab in the ***Controlpoints Configuration*** window is not user friendly. However, the "***Save and Restore on load***" option (also available with breakpoints and watchpoints) can be very useful to automatically save the application with the current DBG module setup and trigger positions just as they were used in the application.

Trigger Settings

The Trigger Settings option of a popup menu can be chosen to set all kinds of triggers without opening the Trigger Module Settings Window on page 580. However, the amount of trigger types is **dynamic**, dependanding if no triggers are defined, if only Trigger A is defined, if only trigger B is defined if both triggers are defined, and also depending on the trigger type (Instruction, Read Access, Read/Write Access, Write Access. **Only possible combinations are displayed.**

Also DBG Module Options on page 591 can be directly edited.

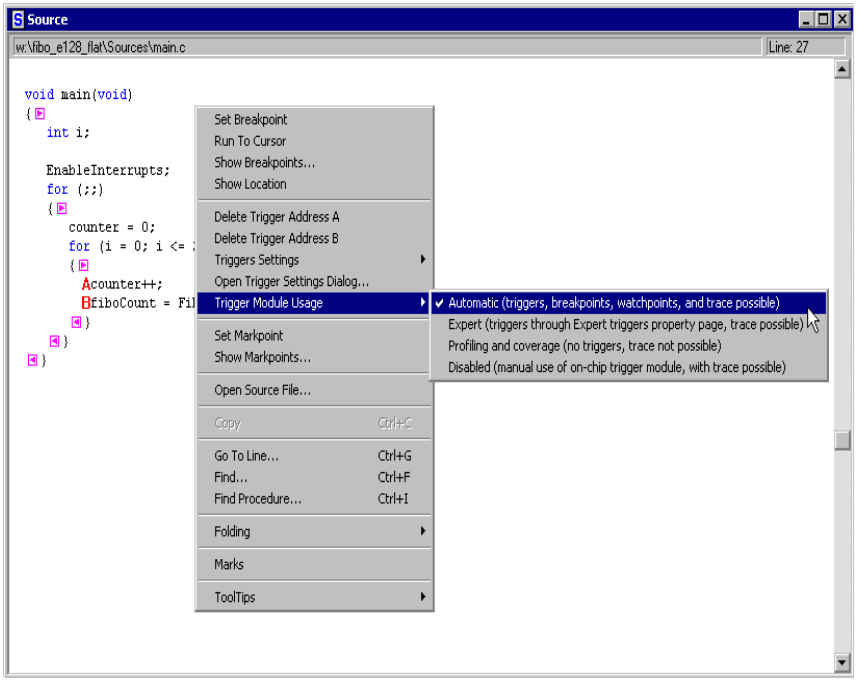
Figure 24.11 Triggers Setting Menu Option - Extended Menu


Set Breakpoint
Run To Cursor
Show Breakpoints...
Show Location
Delete Trigger Address A
Delete Trigger Address B
Triggers Settings
Open Trigger Settings Dialog...
Trigger Module Usage
Set Markpoint
Show Markpoints...
Open Source File...
Copy Ctrl+C
Go To Line... Ctrl+G
Find... Ctrl+F
Find Procedure... Ctrl+I
Folding
Marks
ToolTips
Automatic (triggers, breakpoints, watchpoints, and trace possible)
Instruction at Address A or Address B is executed
Instruction execution inside Address A - Address B range
Instruction execution outside Address A - Address B range
Instruction at Address A then at Address B were executed
Instruction at Address A and value on data bus match
Instruction at Address A and value on data bus mismatch
Record continuously and halt on trigger hit
Record continuously and DO NOT halt on trigger hit
Start recording after trigger hit and halt when fifo is full
Start recording after trigger hit and DO NOT halt when fifo is full

Trigger Module Usage

This menu entry can be used to set globally the DBG module functionality without opening the Trigger Module Settings Window on page 580 to do the DBG Module Mode Setup on page 581.

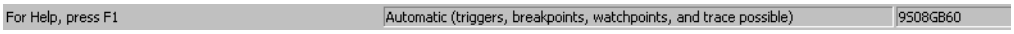
Figure 24.12 Source Window Extended Menu



DBG Support Status Bar Item

A specific DBG support debugger status bar item is present as soon as the debugger target processor features the DBG module. Clicking on this item opens immediately the Trigger Module Settings Window on page 580 (future debugger revision only).

Figure 24.13 Status Bar Item



The status bar displays the current DBG Module Mode Setup on page 581 (as shown above) or the current preset "Instruction" Triggers on page 587, "Memory Access"

Triggers on page 585 or "Capture" Triggers on page 590 used or the current DBG Module Mode Setup on page 581.

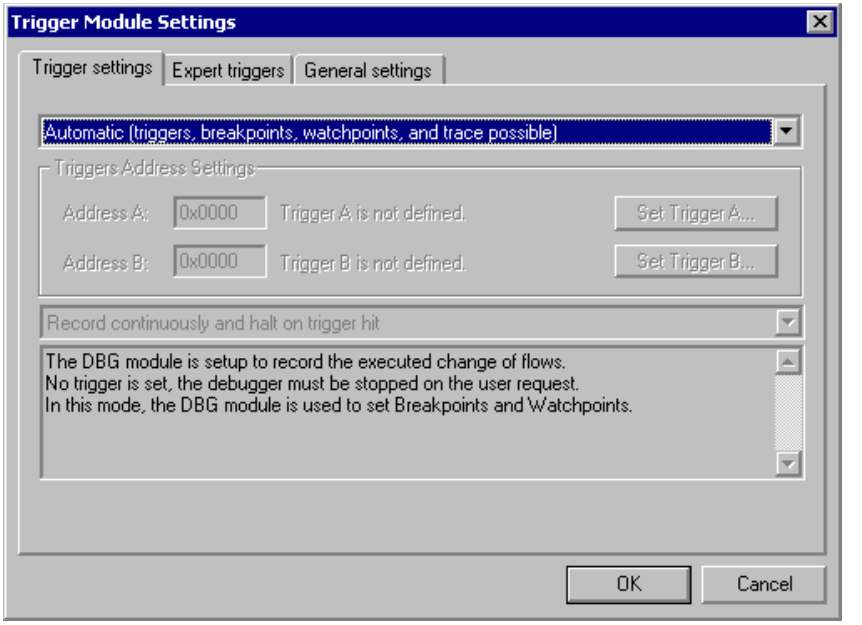
Figure 24.14 Status Bar Item

Trigger Module Settings Window

This window can be opened from context sensitive popup menus in the Source, Data, Memory and Assembly component windows, from the Connection menu and also when clicking on a Status Bar item (future debugger revision only).

The on-chip DBG module can be fully controlled from within this window.

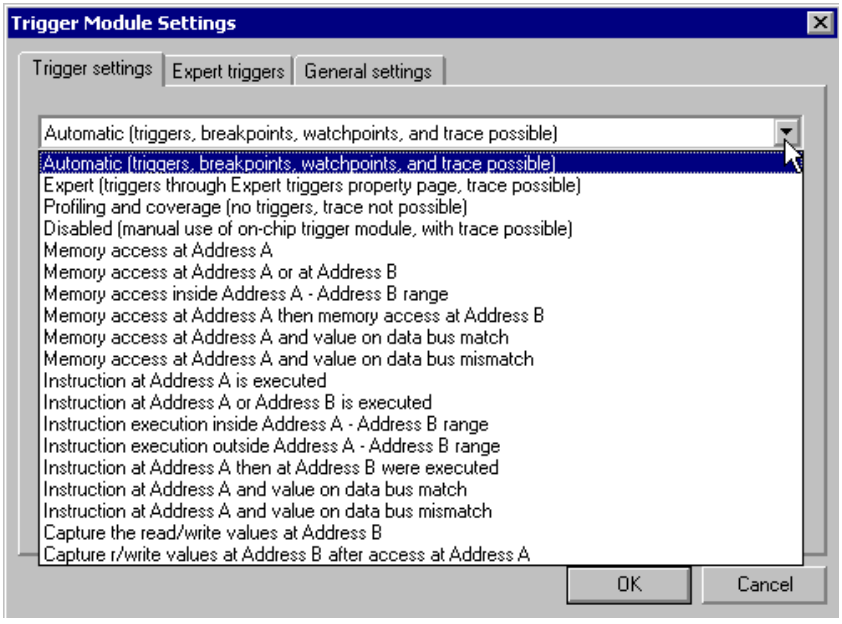
Figure 24.15 Trigger Module Settings Window - Trigger Settings Tab


Trigger Module Settings
Trigger settings
Expert triggers
General settings
Automatic (triggers, breakpoints, watchpoints, and trace possible)
Triggers Address Settings
Address A:
0x0000
Trigger A is not defined.
Set Trigger A...
Address B:
0x0000
Trigger B is not defined.
Set Trigger B...
Record continuously and halt on trigger hit
The DBG module is setup to record the executed change of flows.
No trigger is set, the debugger must be stopped on the user request.
In this mode, the DBG module is used to set Breakpoints and Watchpoints.
OK
Cancel

DBG Module Mode Setup

First of all, the on-chip DBG module provides some exclusive debugging features. Open the top drop down list to display all modes and complex breakpoints/wacthpoints i.e. kind of triggers available.

Figure 24.16 Trigger Settings Tab Listbox


Trigger Module Settings
Trigger settings
Expert triggers
General settings
Automatic (triggers, breakpoints, watchpoints, and trace possible)
Automatic (triggers, breakpoints, watchpoints, and trace possible)
Expert (triggers through Expert triggers property page, trace possible)
Profiling and coverage (no triggers, trace not possible)
Disabled (manual use of on-chip trigger module, with trace possible)
Memory access at Address A
Memory access at Address A or at Address B
Memory access inside Address A - Address B range
Memory access at Address A then memory access at Address B
Memory access at Address A and value on data bus match
Memory access at Address A and value on data bus mismatch
Instruction at Address A is executed
Instruction at Address A or Address B is executed
Instruction execution inside Address A - Address B range
Instruction execution outside Address A - Address B range
Instruction at Address A then at Address B were executed
Instruction at Address A and value on data bus match
Instruction at Address A and value on data bus mismatch
Capture the read/write values at Address B
Capture r/write values at Address B after access at Address A
OK
Cancel

Automatic Mode (Default)

The DBG Module is used to set up **three** hardware breakpoints or one watchpoint or to set up triggers selected by the user from the list or from a context sensitive popup menu. This mode is simply the default selection when no triggers have been set yet.

The trigger condition and trigger addresses can be set from the debugger Source, Assembly, Memory and Data component using Set Trigger A or Set Trigger B context sensitive pop up menu entry,or within this dialog.

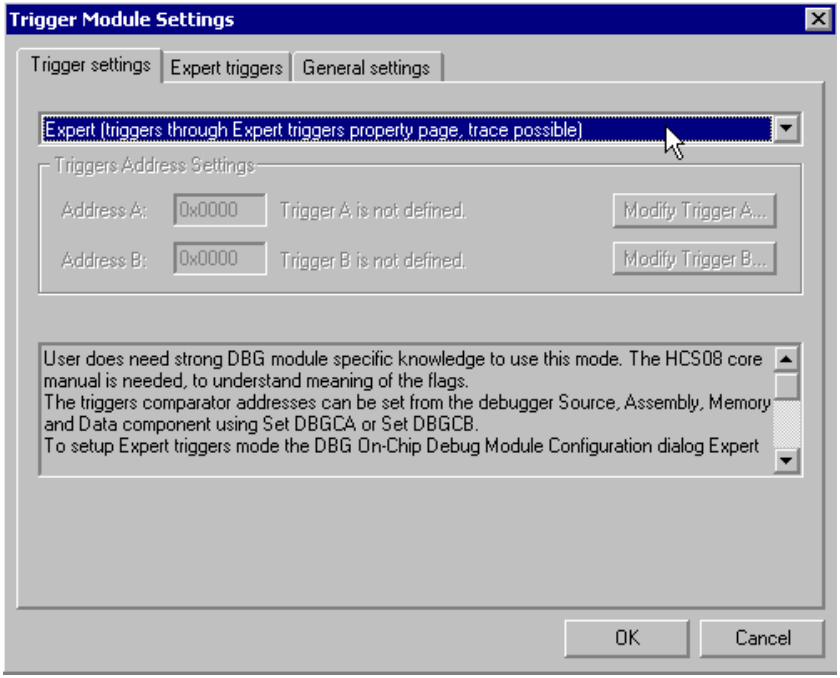
The DBG module is setup to record the executed change of flows. As no triggers are set, the debugger is stopped on the user request or typical breakpoints/watchpoint. To summerize, in this mode, the DBG module is used to set regular hardware breakpoints and watchpoints.

Expert Mode

The User needs to know the on-chip DBG module really well to use this mode. It can be seen as a “Do It Yourself” way to set the DBG module. The HCS08 core manual is needed, to understand the meaning of the registers and flags.

The triggers comparator addresses can be set from the debugger Source, Assembly, Memory and Data windows using Set DBGCA or Set DBGCB. The DBG module is set by the debugger. DBG module enabling and arming depend on the selected flags set within the DBG register control registers (through the Expert triggers tab property page). The settings are written to the hardware right before the application is run. The DBG module is reset when the application stops.

Figure 24.17 Trigger Setings Tab - Expert Mode Information


Trigger Module Settings
Trigger settings
Expert triggers
General settings
Expert (triggers through Expert triggers property page, trace possible)
Triggers Address Settings
Address A:
0x0000
Trigger A is not defined.
Modify Trigger A...
Address B:
0x0000
Trigger B is not defined.
Modify Trigger B...
User does need strong DBG module specific knowledge to use this mode. The HCS08 core manual is needed, to understand meaning of the flags.
The triggers comparator addresses can be set from the debugger Source, Assembly, Memory and Data component using Set DBGCA or Set DBGCB.
To setup Expert triggers mode the DBG On-Chip Debug Module Configuration dialog Expert
OK
Cancel

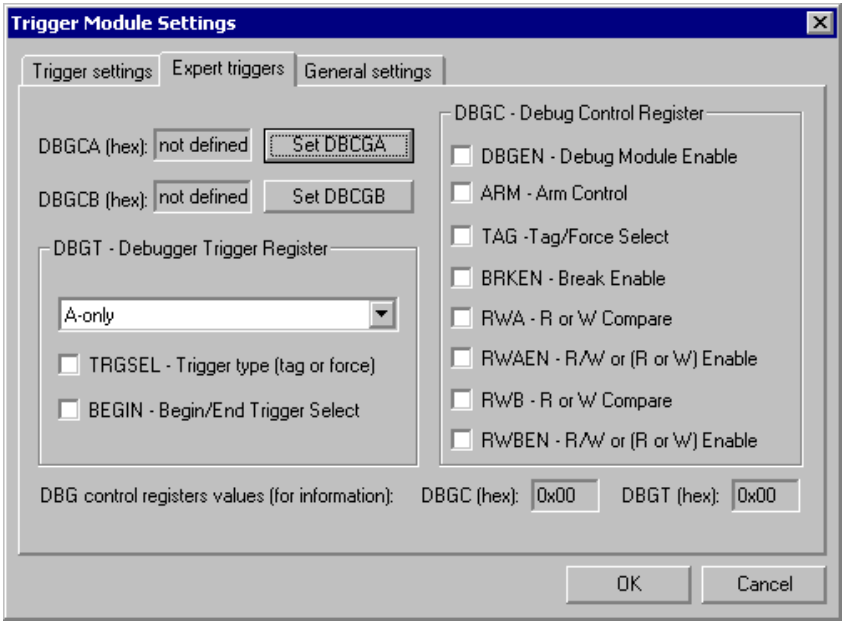
To set Expert triggers , the Trigger Module Settings window ***“ExpertTtriggers”*** tab must be used. Select the ***Expert*** mode in the drop down list to enable the Expert Mode Tab on page 583.

Expert Mode Tab

The expert mode tab gives you an access to most of the on-chip DBG module registers. Trigger types can be directly set from the "***DBGT - Debugger Trigger Register***" drop down list.

Code program flow rebuild and data recording are also synchronized with the Expert mode and results are displayed in theTrace Component Window on page 597.

Figure 24.18 Trigger Module Settings Window - Expert Triggers Tab



Profiling and Coverage Mode

Choosing this mode, the DBG module is setup to source code execution profiling and source code execution coverage . The Profiler and/or Coverage components should be opened to display results.

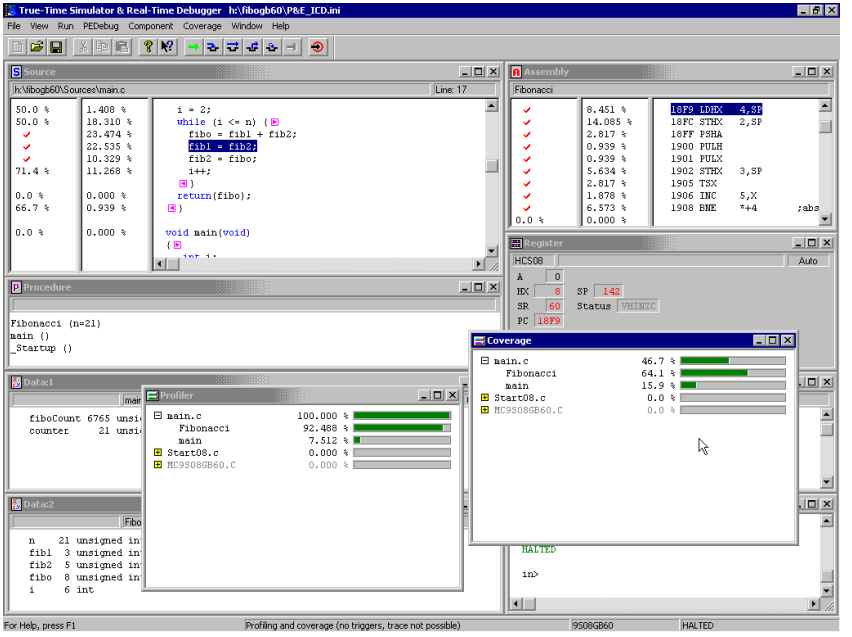
Neither triggers nor DBG based controlpoints can be set in this mode, and the debugger must be stopped on the user request (software breakpoints can still be used).

Profiling and Coverage features are based on a periodical debugger program counter real time fetch from the debugger to the on-chip DBG module. Also this fetch is statistical and cannot cover all program counters and longer is the program running and tests period, more precise would get resulted statistics.

Please see also Limitations on page 604 section for this mode.

Please refer to the debugger engine manual for Coverage and Profiler component features.

Figure 24.19 Debugger Main Window - Coverage and Profiler Windows



Disabled Mode

The User needs to know the on-chip DBG module to use this mode. It can be seen as a “Do It Yourself” way to set hardware breakpoints, watchpoints, and triggers. Please consult the Reference Document on page 567 section and documents to get all infomation about the HCS08 on-chip DBG module before attempting to use this mode.

There is no dedicated graphical user interface to access DBG module register. The triggers comparator addresses and DBG control registers are handled by the user through the debugger Memory component or using command line commands. The DBG module is NOT set by the debugger. DBG module enabling and arming depend on the selected flags set within the DBG register control registers. The DBG module is NOT reset when the application stops. By default, the FIFO content is protected from unexpected reads, the DBG module is automatically disarmed and the FIFO is analyzed when the debugger stops. This can be optionally disabled by the user.

“Memory Access” Triggers

Memory Access at Address A

This mode is used to trigger on a program instruction read and/or write at Address A memory location.

The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Memory Access at Address A or Address B

This mode is used to trigger on a program instruction read and/or write at Address A or at Address B memory location.

The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Memory Access Inside Address A - Address B Range

This mode is used to trigger on a program instruction read and/or write inside the Address A - Address B memory range locations.

The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Memory Access at Address A then Memory Access at Address B

This mode is used to trigger on a program instruction sequence first reading and/or writing at Address A memory location then reading and/or writing at Address B memory location.

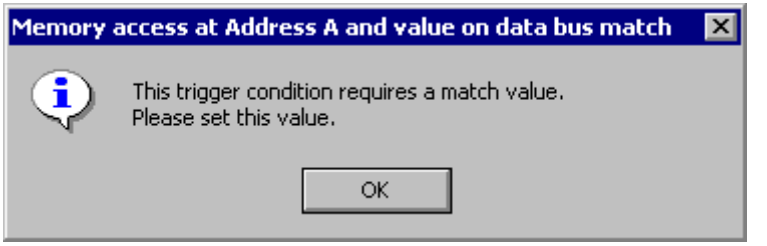
The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Memory Access at Address A and Value on Data Bus Match

This mode is used to trigger on a program instruction read and/or write of a specific matching byte value at Address A memory location.

When choosing this trigger type, the **trigger B** address is used as a **match value** rather than an address. Also when setting this trigger via a context sensitive popup menu, the following message is displayed if the match value was never set.

Figure 24.20 Memory Access at Address A and Value on Data Bus Match Dialog Box


Memory access at Address A and value on data bus match
This trigger condition requires a match value.
Please set this value.
OK

The Trigger Editing on page 593 dialog is not available for the trigger B. Special "***Match value***" edit boxes are displayed instead of ***Address B*** edit box.

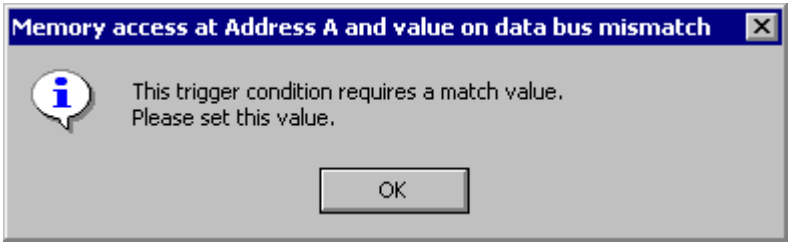
The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Memory Access at Address A and Value on Data Bus Mismatch

This mode is used to trigger on a program instruction read and/or write of a NOT matching byte value at Address A memory location.

When choosing this trigger type, the **trigger B** address is used as a **mismatch value** rather than an address. Also when setting this trigger via a context sensitive popup menu, the following message is displayed if the match value was never set.

Figure 24.21 Memory Access at Address A and Value on Data Bus Mismatch Dialog Box


Memory access at Address A and value on data bus mismatch
This trigger condition requires a match value.
Please set this value.
OK

The Trigger Editing on page 593 dialog is not available for the trigger B. Special "***Match value***" edit boxes are displayed instead of ***Address B*** edit box.

The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

"Instruction" Triggers

Instruction at Address A Is Executed

This mode is used to trigger on a program instruction execution (program counter) at Address A.

The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Instruction at Address A or Address B Is Executed

This mode is used to trigger on a program instruction execution (program counter) at Address A or at Address B.

The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Instruction Execution Inside Address A - Address B Range"

This mode is used to trigger on a program instruction execution (program counter) inside the Address A - Address B range.

The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Instruction Execution Outside Address A - Address B Range"

This mode is used to trigger on a program instruction execution (program counter) outside the Address A - Address B range.

NOTE **IMPORTANT**: With the **HCS08 Serial Monitor via GDI connection**, this trigger type might be be interferedwith by the monitor code intself and therefore the debugger might break for executed code not belonging to the user application.

The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Instruction at Address A then at Address B Were Executed

This mode is used to trigger on a program instruction execution (program counter) sequence first at Address A then at Address B.

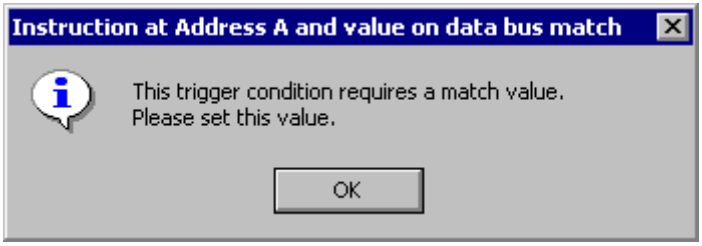
The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Instruction at Address A and Value on Data Bus Match

This mode is used to trigger on a program instruction execution (program counter) at Address A, this instruction opcode matching a specific byte value.

When choosing this trigger type, the **trigger B** address is used as a **match value** rather than an address. Also when setting this trigger via a context sensitive popup menu, the following message is displayed if the match value was never set.

Figure 24.22 Memory Access at Address A and Value on Data Bus Match Dialog Box


Instruction at Address A and value on data bus match
This trigger condition requires a match value.
Please set this value.
OK

The Trigger Editing on page 593 dialog is not available for the trigger B. Special "***Match value***" edit boxes are displayed instead of ***Address B*** edit box.

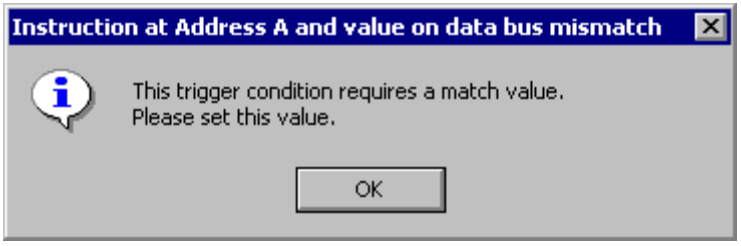
The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

Instruction at Address A and Value on Data Bus Mismatch

This mode is used to trigger on a program instruction execution (program counter) at Address A, this instruction opcode NOT matching a specific byte value.

When choosing this trigger type, the **trigger B** address is used as a **mismatch value** rather than an address. Also when setting this trigger via a context sensitive popup menu, the following message is displayed if the match value was never set.

Figure 24.23 Memory Access at Address A and Value on Data Bus Mismatch Dialog Box


Instruction at Address A and value on data bus mismatch
This trigger condition requires a match value.
Please set this value.
OK

The Trigger Editing on page 593 dialog is not available for the trigger B. Special "***Match value***" edit boxes are displayed instead of ***Address B*** edit box.

The code program flow rebuild is displayed in the Trace Component Window on page 597 automatically switched to on page 603Instructions Display on page 598 mode.

"Capture" Triggers

Capture Read/Write Values at Address B

This mode is used to capture the data involved in a read and/or write access to the address specified by the **trigger B**, such as the address of a particular control register or program variable.

Captured byte data are displayed in the Trace Component Window on page 597 automatically switched to on page 603Recorded Data Display on page 603 mode.

The trigger address is typically not a program code address (program counter), but rather a data/memory address.

Capture Read/Write Values at Address B After Access at Address A

This mode is used to capture the data involved in a read and/or write access to the addresses specified by the **trigger A** and the **trigger B**, such as the address of a particular control register or program variable. Triggering/capture will start only after the **trigger A** address was accessed.

The trigger addresses is typically not a program code address (program counter), but rather data/memory addresses.

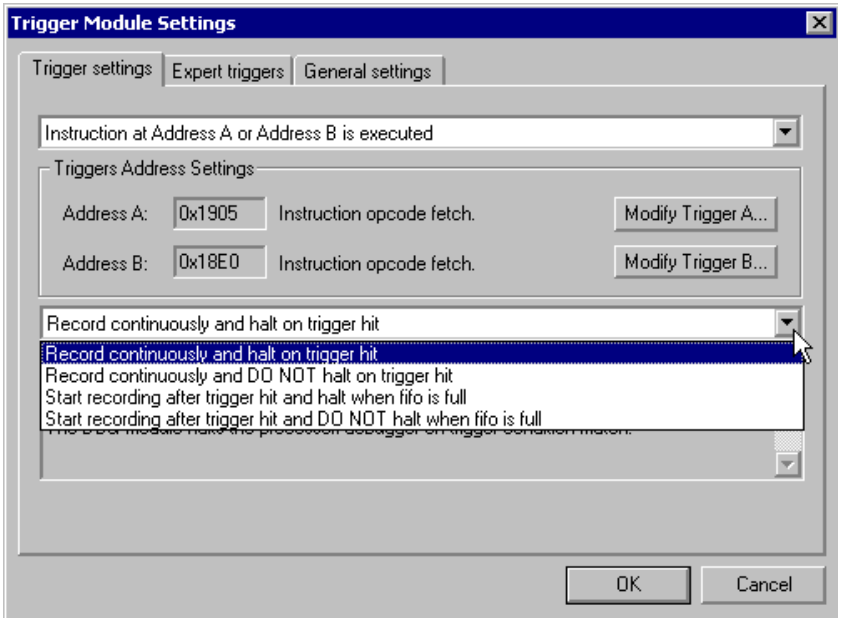
Captured byte data are displayed in the Trace Component Window on page 597 automatically switched to on page 603Recorded Data Display on page 603 mode.

DBG Module Options

Program Code Change of Flow Recording

The program code change of flow options are available for "Instruction" Triggers on page 587 and "Memory Access" Triggers on page 585 and controlled through the Trigger Module Settings window's Trigger Settings tab list box.

Figure 24.24 Change of Flow Recording Control


Trigger Module Settings
Trigger settings
Expert triggers
General settings
Instruction at Address A or Address B is executed
Triggers Address Settings
Address A:
0x1905
Instruction opcode fetch.
Modify Trigger A...
Address B:
0x18E0
Instruction opcode fetch.
Modify Trigger B...
Record continuously and halt on trigger hit
Record continuously and halt on trigger hit
Record continuously and DO NOT halt on trigger hit
Start recording after trigger hit and halt when fifo is full
Start recording after trigger hit and DO NOT halt when fifo is full
OK
Cancel

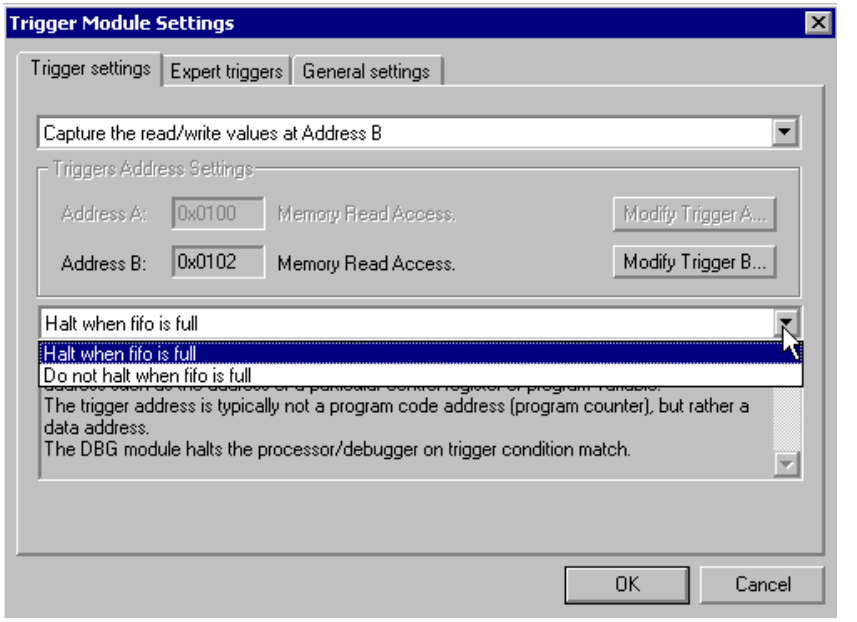
- ***Record continuously and halt on trigger hit***: The DBG module starts recording program flow information immediately after run. The DBG module halts the processor/debugger on trigger condition match.
- ***Record continuously and DO NOT halt on trigger hit***: The DBG module starts recording program flow information immediately after run. The DBG module does not halt the processor/debugger on trigger condition match.
- ***Start recording after trigger hit and halt when the fifo is full***: The DBG module starts recording program flow information on trigger condition match and halts the processor/debugger when the capture buffer is full.

- ***Start recording after trigger hit and halt when the fifo is full***: The DBG module starts recording program flow information on trigger condition match. The DBG module does not halt the processor/debugger on trigger condition match.

When Data Recording

The data recording options are available for "Capture" Triggers on page 590 only and are selected from the list box in the Trigger Settings tab of the Trigger Module Settings window.

Figure 24.25 Data Recording Control

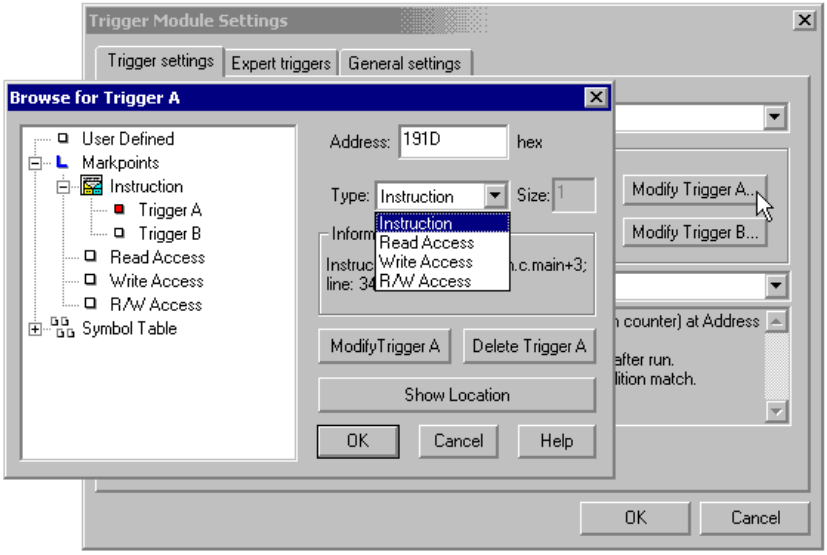

Trigger Module Settings
Trigger settings
Expert triggers
General settings
Capture the read/write values at Address B
Triggers Address Settings
Address A:
0x0100
Memory Read Access.
Modify Trigger A...
Address B:
0x0102
Memory Read Access.
Modify Trigger B...
Halt when fifo is full
Halt when fifo is full
Do not halt when fifo is full
The trigger address is typically not a program code address (program counter), but rather a data address.
The DBG module halts the processor/debugger on trigger condition match.
OK
Cancel

- ***Halt when the fifo is full***: The DBG module records continuously data accesses and halts the processor/debugger when the capture buffer is full.
- ***Do not halt when the fifo is full***: The DBG module records continuously data accesses and but does not halt the processor/debugger when the capture buffer is full.

Trigger Editing

Typically trigger addresses and/or typecan be set using context sensitive popup menus. It is also possible to modify trigger addresses and type within the Trigger Module Settings Window on page 580. Pressing "***Modify Trigger***" buttons opens a trigger editor dialog box.

Figure 24.26 Browse for Trigger A Dialog Box



In the trigger editor dialog box:

The "***Address***" edit box contains the inital and final trigger address value. This value can be directy set by typing in the edit box.

The "***Type***" drop down list should be used to select/change the type of trigger. "Instruction" type should be used for "Instruction" Triggers on page 587 and "***Read***", "***Write***" and "***R/W Access***" should be used for"Memory Access" Triggers on page 585 and "Capture" Triggers on page 590.

Pressing "***Modify Trigger***" in this sub dialog will modify and record the trigger in the trigger database (Trigger Storing as Markpoints on page 571).

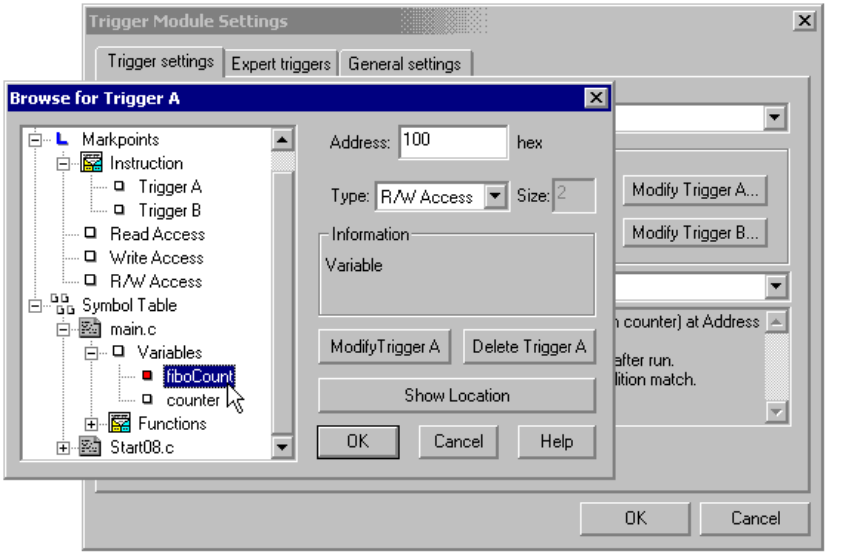
NOTE Pressing the OK button does NOT update the trigger database. The"***Modify Trigger***" button in the Trigger Module Settings window must be explicitly pressed before closing the dialog box to update the trigger database.

Pressing "***Delete Trigger***" in the dialog box removes the trigger in the trigger database (Trigger Storing as Markpoints on page 571). This trigger address is then considered as "**undefined**".

The "***Show Location***" button shows the location of the trigger (as program code location or program data) in the Source, Data, Assembly and Memory windows .

The left hand side tree is a user friendly way to find a trigger address in the debugger symbol database by selecting a variable (the address of the variable will be taken and copied in the ***Address*** edit box) or a function (the entry point of the function will be taken and copied in the ***Address*** edit box) , and also regular markpoints (the address of the markpoint will be taken and copied in the ***Address*** edit box) from the markpoint list.

Figure 24.27 Finding Trigger Address in Editor Dialog Box



Trigger Module Settings Window - Display Information

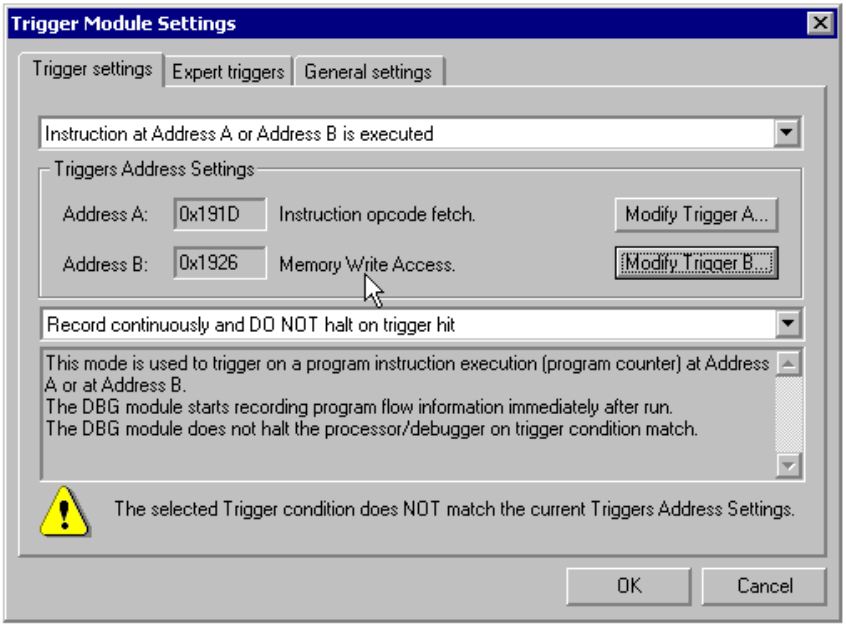
A large grayed edit box dynamically provides information about the current triggers and selected options.

As context sensitive popup menus will only display triggers matching the amount and the kind of triggers which are currently set, the Trigger Module Settings Window on page 580 on page 580 checks dynamically the validity are current triggers set vs. the trigger mode.

As shown below, if one or more tirggers do not match the trigger mode selection, a warning icon and message is displayed on the bottom of the dialog.

Here below, the ***Memory Write Access*** type of trigger selected by the mouse cursor does not match with the "Instruction" Triggers on page 587 type selected in the drop down list.

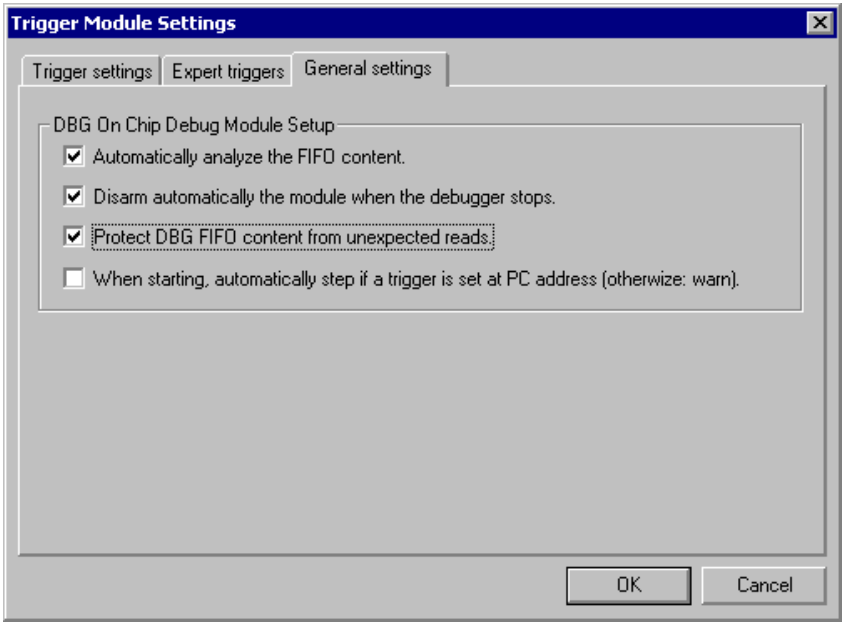
Figure 24.28 Trigger Settigs Tab Information


Trigger Module Settings
Trigger settings
Expert triggers
General settings
Instruction at Address A or Address B is executed
Triggers Address Settings
Address A:
0x191D
Instruction opcode fetch.
Modify Trigger A...
Address B:
0x1926
Memory Write Access.
Modify Trigger B...
Record continuously and DO NOT halt on trigger hit
This mode is used to trigger on a program instruction execution (program counter) at Address A or at Address B.
The DBG module starts recording program flow information immediately after run.
The DBG module does not halt the processor/debugger on trigger condition match.
The selected Trigger condition does NOT match the current Triggers Address Settings.
OK
Cancel

General Settings Tab

Most of the time, there would be no reason to change any of these settings, which are rather default settings of the DBG user interface. However, in some debug special cases, it is possible to disable some automated debugger background processes.

Figure 24.29 Trigger Module Settings Window - General Settings Tab

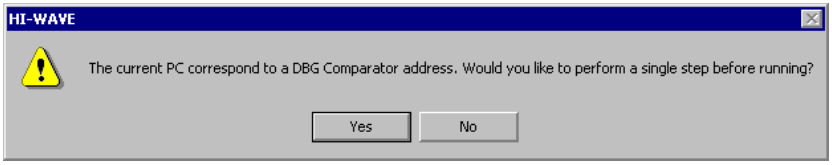

Trigger Module Settings
Trigger settings
Expert triggers
General settings
DBG On Chip Debug Module Setup
Automatically analyze the FIFO content.
Disarm automatically the module when the debugger stops.
Protect DBG FIFO content from unexpected reads.
When starting, automatically step if a trigger is set at PC address (otherwize: warn).
OK
Cancel

- ***Automatically analyze the FIFO content***: When the Trace Component Window on page 597 is open, after the debugger is halted by the user or a breakpoint, watchpoint or a trigger, DBG module results are automatically analysed then displayed in the Trace window. If the Trace window is closed, the DBG user interface does not perform any result analyse except trigger flags reported in the status bar. Unchecking this check box would do the same, with the Trace window open.
- ***Disarm automatically the module when the debugger stops***: By default, once the debugger target processor is halted due to user break (not any trigger), the on-chip DGB module is still armed. If this option is selected (by default) the debugger will disarm it to retrieve data from the DBG Fifo. If not selected, the DBG Fifo/buffer information cannot be retrieved until the module is disarmed.
- ***Protect DBG FIFO content from unexpected reads***: The DBG Fifo data are retrieved from ***DBGFH-DBGFL*** registers (address 0x1814-0x1815 in register block at reset location). Several reads are performed to retrieve the entire shifting buffer.

However, when the debugger is halted, while refreshing Data and Assembly windows, it might read also the debugger target processor memory at the same location, reading the first DBG Fifo data, shifting the buffer, and therefore corrupt the DBG user interface DBG Fifo data retrieving. This option hides to the debugger and also user (see blue "-- --" designs in the Memory window at address 0x1814-0x1815) the DBG Fifo buffer location.

- ***When starting, automatically step if a trigger is set at PC address (otherwize: warn)***: To run again the application, the debugger usually needs to exit the trigger current match condition and avoid being stuck/halted/locked by the trigger. A single step is usually required on page 587to "escape" from "Instruction" Triggers on page 587. When this option is disabled, the debugger prompts the following dialog to validate this choice.

Figure 24.30 Trigger "Escape" Dialog Box


HI-WAVE
The current PC correspond to a DBG Comparator address. Would you like to perform a single step before running?
Yes
No

Trace Component Window

The Trace component is a debugger generic component used to display in a Trace window a debugger internal database. The context sensitive popup menu is set up by the connection (or the GDI DLL) making usage of the component.

Any debugger connections including the DBG user interface are synchronized with the Trace component.

It is not necessary to open the Trace window/component to make usage of the DBG user interface triggers. However, several triggers are used to collect code program flow information or access data information. The Trace window can be opened from Specific Connection Menu Options on page 568, from Popup Menu Entries in Source, Data, Assembly and Memory Windows on page 568 , and from the DBG Support Status Bar Item on page 579. The window can by saved in the debugger layout when pressing the debugger Save icon.

NOTE When the Trace component/window is closed, the debugger might be faster, as code program flow rebuild is discarded, this last disassembling back the assembly data from the connection CPU's memory.

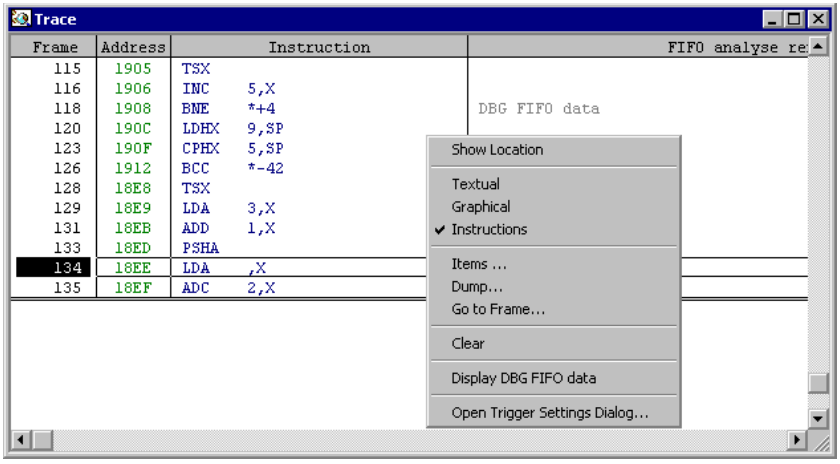
Instructions Display

This display mode is automatically set when "Instruction" Triggers on page 587 and "Memory Access" Triggers on page 585 are used. It is also the default display in Automatic Mode (Default) on page 581.

Displayed columns:

- ***Frame***: A number representing an information item stored in the Trace component database.
- ***Address***: instruction program counter.
- ***Instruction***: code program flow instruction disasssembly.
- ***FIFO Analyse remark***: a debugger information: "***DBG FIFO data***" means that this data was recorded by the on-chip DBG module. "***traced***" means an item/instruction optained by debugger/user singlestep or assembly step. "***Program flow rebuild gap***" means that the debugger could not track completely the code program flow between two frames.

Figure 24.31 Trace Window - Popup Menu Options

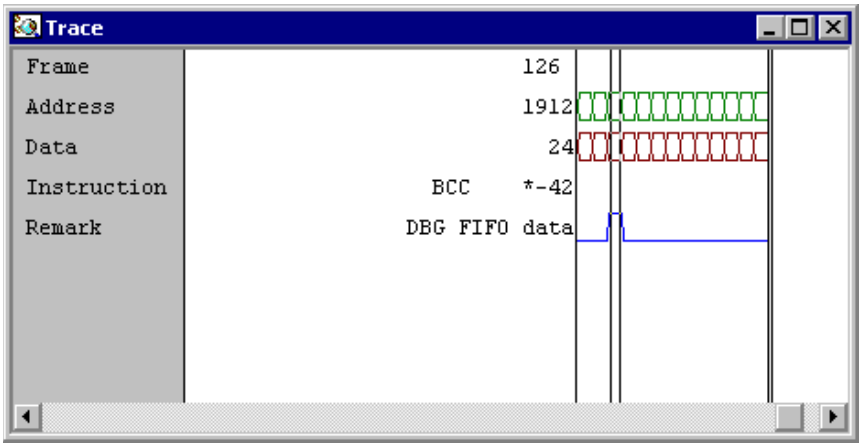


Selecting "***Show Location***" in the Trace window context sensitive popup menu will simply display in Source and Assembly window the frame matching source and assembly code.

Graphical Display

This display mode can be select when selecting "***Graphical***" in the Trace window context sensitive popup menu. It provides a graphical representation of the same information.

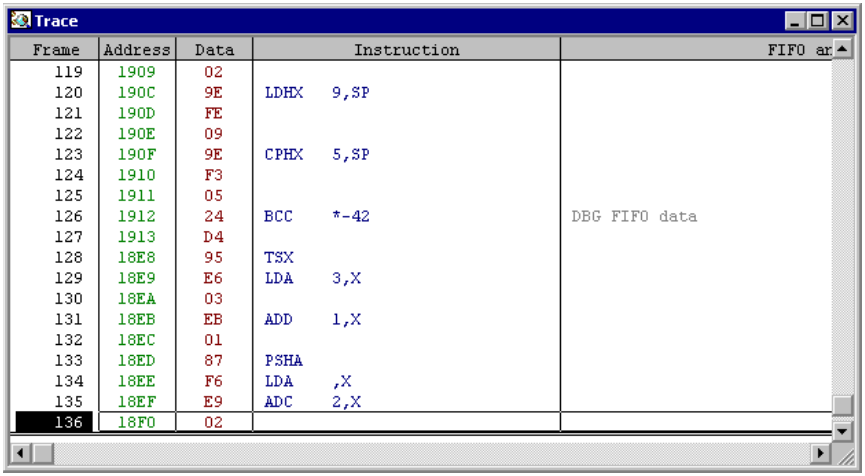
Figure 24.32 Trace Window - Graphical Display



Textual Display

This display mode can be select when selecting "***Textual***" in the Trace window context sensitive popup menu , when using "Instruction" Triggers on page 587 and "Memory Access" Triggers on page 585 are used. This display mode is rather useless for the DBG user interface, as no read/write accesses are recorded at the same time than program change of flow information by the on-chip DBG module. By consequence, the Textual display mode simply expands instruction assembly code in the Trace window.

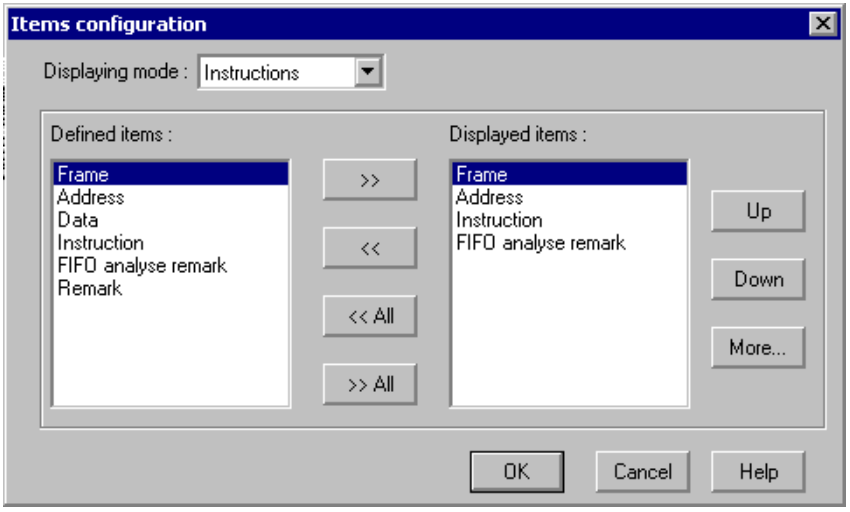
Figure 24.33 Trace Window - Textual Display



Column Display and Moving

Selecting "***Items...***" in the Trace window context sensitive popup menu opens a small dialog to setup the columns to hide/display in each display mode. The "Displaying mode" drop down list can be opened to make column display modification in *Textual*, *Instructions* or *Graphical* mode.

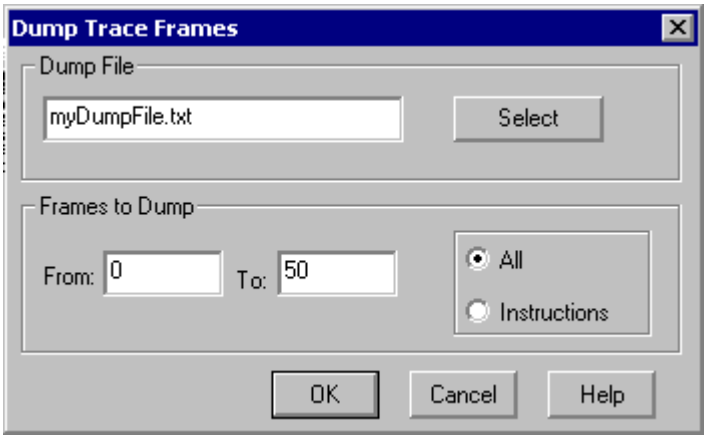
Figure 24.34 Items Configuration Dialog Box



Dumping Frames to File

Selecting "***Dump...***" in the Trace window contest sensitive popup menu opens a small dialog to dump/save Trace component frames to a text file.

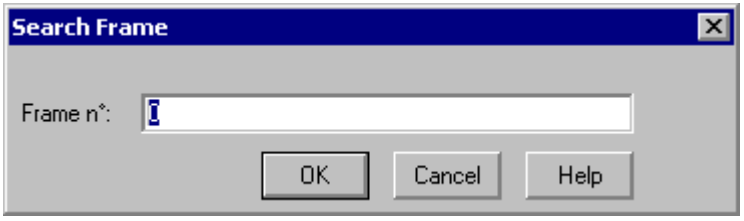
Figure 24.35 Dump Trace Frames Dialog Box



Goto Frame

Selecting "***Go to Frame...***" in the Trace window context sensitive popup menu opens a small dialog to go to a frame in the Trace window.

Figure 24.36 Search Frames Dialog Box



Clearing Frames

Selecting "***Clear***" in the Trace window context sensitive popup menu will simply flush the frames in the Trace window (flushing in background the database).

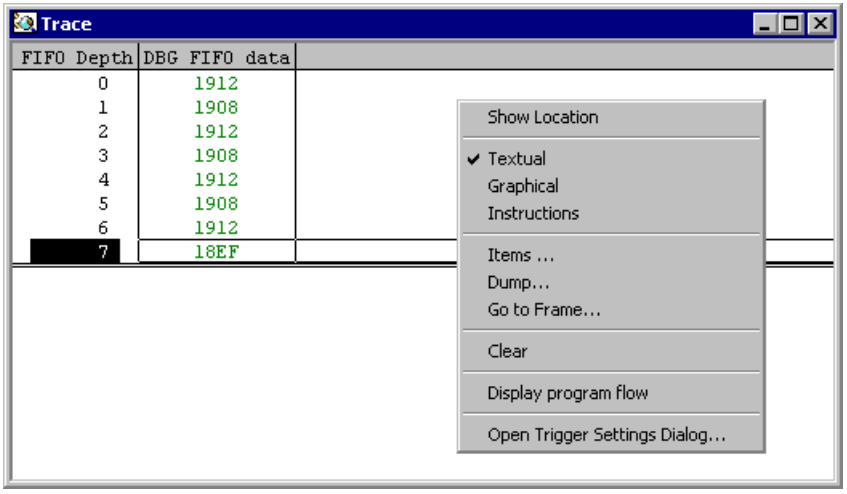
DBG Module FIFO/Buffer Display

Selecting "***Display DBG FIFO data***" in the Trace window contest sensitive popup menu will simply display data information retrieved from the on-chip DGB module Fifo/buffer. Selecting "***Display program flow***" in the Trace window context sensitive popup menu will simply turn back to code program flow display.

Displayed columns:

- ***FIFO Depth***: A number representing the depth in the DBG/Fifo of the word data value. The first frame (Depth 1) is the oldest value in the time.
- ***DBG FIFO Data***: the word value retrieved from the DBG Fifo/buffer from DBGFH and DBGFL DBG on-chip module registers.

Figure 24.37 Trace Window - FIFO Display



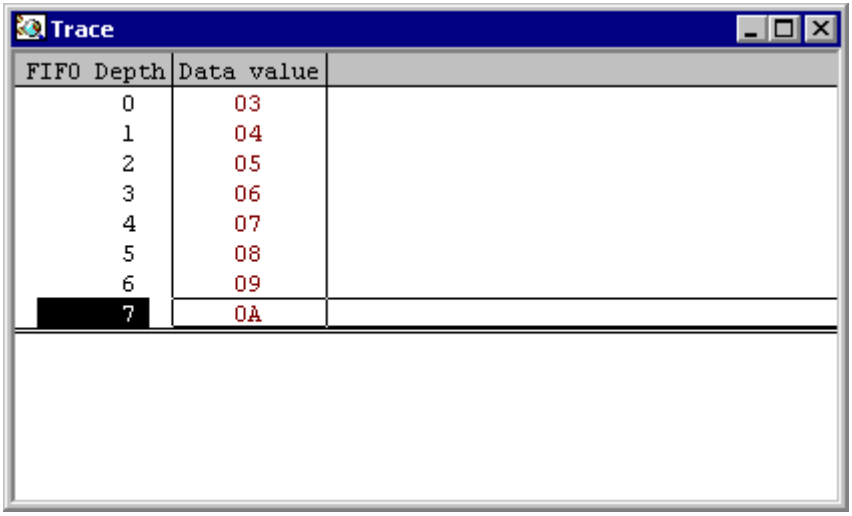
Recorded Data Display

This display mode is automatically set when "Capture" Triggers on page 590 are used.

Displayed columns:

- ***FIFO Depth***: A number representing the depth in the DBG/Fifo of the byte data value. The first frame (Depth 1) is the oldest value in the time.
- ***Data value***: the byte value retrieved from the DBG Fifo/buffer from the DBGFL DBG on-chip module register.

Figure 24.38 Trace Window - Recorded Data Display



Limitations

Demo/unregistered debugger mode

- In demo/unregistered debugger mode, code program reconstruction has a limited number of frames displayed in the Trace window.
- Also Real time code Profiling and code Coverage are disabled.
- No preset/predefined "Instruction" Triggers on page 587, "Memory Access" Triggers on page 585 or "Capture" Triggers on page 590 are provided. Only Expert Triggers on page 575 can be set.

Book IV - RS08 Debugger Connections

Book IV Contents

Each section of the Debugger manual includes information to help you become more familiar with the Debugger, to use all its functions and help you understand how to use the environment. This book, the RS08 Debugger Connections Common Features, defines the connections available for debugging code written for RS08 CPUs.

This book consists of the following sections:

- "RS08 Full Chip Simulation" on page 607
- "RS08 P&E Multilink/Cyclone PRO Connection" on page 613
- "SofTec RS08 Connection" on page 623

25

RS08 Full Chip Simulation

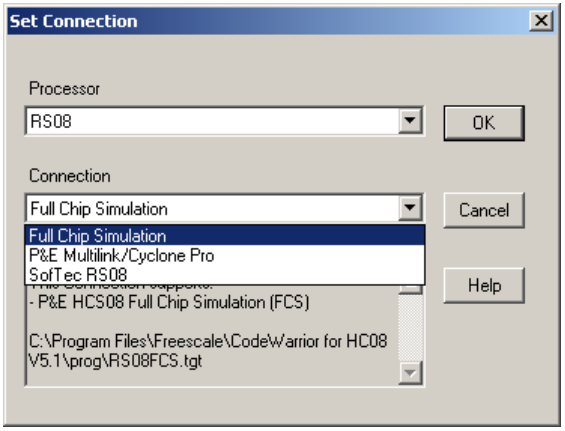
Full Chip Simulation (FCS) does not involve real input and output. Because of this, it does not require a target device to be connected to your PC. The RS08FCS connection simulates the execution of code on the user's MCU system, including the function of any peripherals associated with the device that you select. For more detailed information, refer to the Full Chip Simulation description for the module that you are using.

Configuration Procedure

To select Full Chip Simulation as the debugger connection:

1. Choose the Full Chip Simulation option from the set connection dialog box. See Figure 25.1 on page 607.
2. Click the OK button.

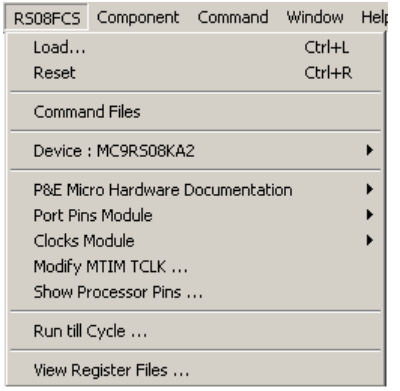
Figure 25.1 Set Connection Dialog Box



Connection (RS08FCS) Menu

Once you have chosen Full Chip Simulation as your debugger connection, the name of the Connection menu is updated and additional options are added.

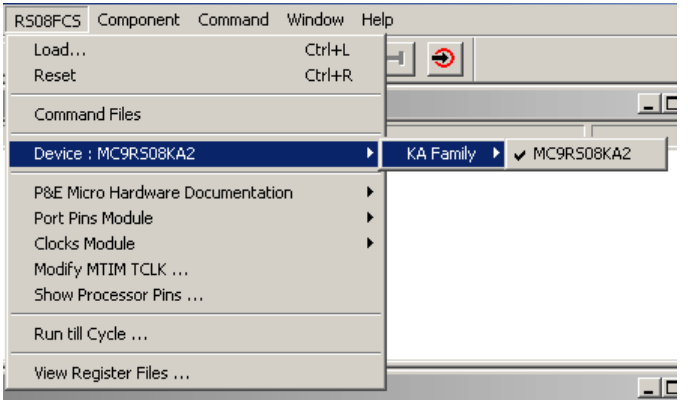
Figure 25.2 RS08FCS Menu



Device Option

The Device selection of the RS08FCS menu allows the user to select the particular Freescale processor that they wish to use. When choosing the Device option from the RS08FCS menu, extended menus will open which allow you to select the family (e.g. KA Family), and device type (e.g. MC9RS08KA2) of the MCU that you are using.

Figure 25.3 RS08FCS Device Extended Menus

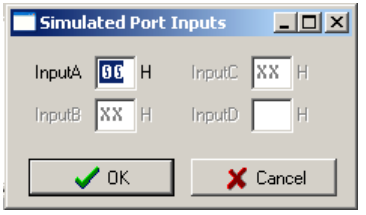


Full Chip Simulation Module Commands

The RS08FCS Menu contains the Full Chip Simulation commands for the modules that have specialty commands associated with them for a chosen device. For more information about specific module commands please refer to the Full Chip Simulation section describing the module.

Port Pins Module

Figure 25.4 Simulated Port Inputs Dialog

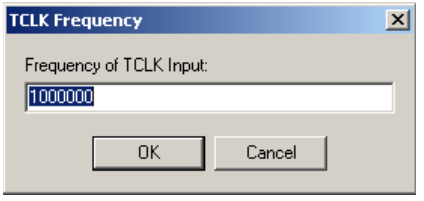


The port pins module menu contains the option to show the input pin levels dialog (this corresponds to the INPUTS command). The dialog graphically displays the current value of the inputs buffers for all available I/O ports on the device (please note that this may not reflect the value shown in the memory window, depending on the current state of a given port's data direction register).

The input buffer can be directly modified from this dialog.

Modify MTIM TCLK

Figure 25.5 TCLK Frequency Dialog



This dialog allows the user to set the frequency of the TCLK signal for the MTIM peripheral. In order for this value to have any effect, the TCLK must be selected as the clock source for the MTIM.

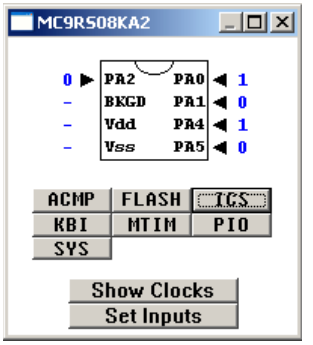
Show Processor Pins

The Processor Pins Dialog is a convenient tool for monitoring the current state of the processor pins, as well as the peripheral with which a pin may be associated. The processor is represented, in graphical form, in the middle of the dialog. Each "pin" is

labeled with its current function. For example, Pin 2 will be the BKGD pin upon reset. However, with the appropriate write to the SOPT register, the user can observe that Pin 2 will change to “PTA3” in the Processor Pins Dialog.

In the dialog, each processor pin (with the exception of the power pins, Vdd and GND) has a corresponding arrow. If the arrow points towards the processor, this indicates that the pin is configured as an input. Conversely, if the arrow is pointing away from the processor, this indicates that the pin is configured as an output.

Figure 25.6 Processor Pins Window

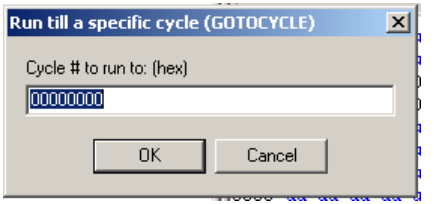


Below the graphical processor representation are several buttons, each corresponding to a processor peripheral. Pressing the button of a given peripheral will bring up the appropriate register files (see below), allowing for easy and informative manipulation of all status, control, and data bits associated with a peripheral.

At the bottom of the dialog are two additional buttons. The “Show Clocks” button will bring up the simulated clock frequencies. The “Set Inputs” button will bring up the Inputs Dialog, which will allow the user to set the simulated input buffers to any valid value.

Run Till Cycle Command

Figure 25.7 Run Till A Specific Cycle Dialog

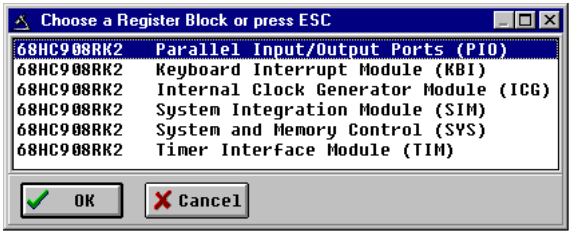


This menu option, which corresponds to the GOTOCYCLE command, will bring up an input dialog which allows the user to specify a given cycle value. When the master cycle counter for the simulator reaches the input value, simulation will halt.

View Register Files Command

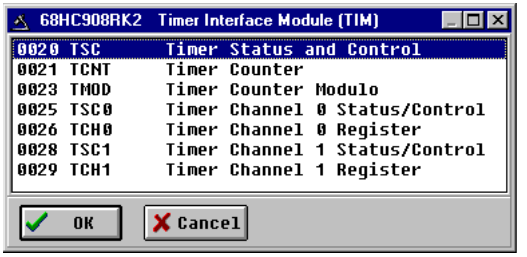
The “View Register Files...” selection in the RS08FCS menu also gives the user the option of running the register file viewer/editor. If register files are available for the device that you have chosen, the “Choose a Register Block” window (see Figure 25.8 on page 611) will be opened. You may also open it by entering the R command in the Command Window command line.

Figure 25.8 Choose A Register Block Dialog Box


Choose a Register Block or press ESC
68HC908RK2 Parallel Input/Output Ports (PIO)
68HC908RK2 Keyboard Interrupt Module (KBI)
68HC908RK2 Internal Clock Generator Module (ICG)
68HC908RK2 System Integration Module (SIM)
68HC908RK2 System and Memory Control (SYS)
68HC908RK2 Timer Interface Module (TIM)
OK
Cancel

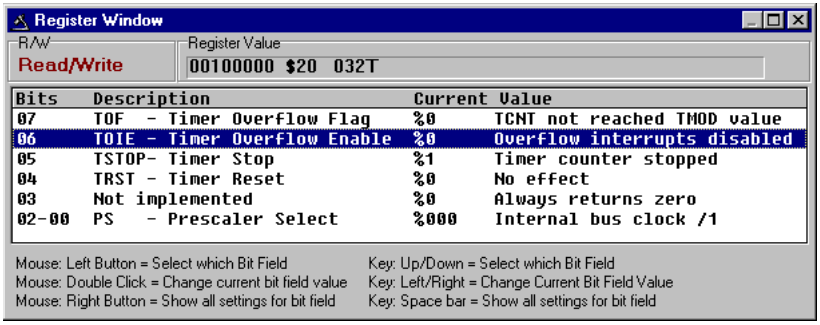
If register files have been installed on the host computer, selecting a block brings up the Register Block register listing (see Figure 25.9 on page 611), which shows a list of the files, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.

Figure 25.9 Timer Interface Module Register Listing


68HC908RK2 Timer Interface Module (TIM)
0020 TSC Timer Status and Control
0021 TCNT Timer Counter
0023 TMOD Timer Counter Modulo
0025 TSC0 Timer Channel 0 Status/Control
0026 TCH0 Timer Channel 0 Register
0028 TSC1 Timer Channel 1 Status/Control
0029 TCH1 Timer Channel 1 Register
OK
Cancel

Selecting a file brings up the Register Window (see Figure 25.10 on page 612), which displays the value and significance of each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.

Figure 25.10 Register Window


Register Window
R/W
Read/Write
Register Value
00100000 $20 032T
Bits Description Current Value
07 TOF - Timer Overflow Flag %0 TCNT not reached TMOD value
06 TOIE - Timer Overflow Enable %0 Overflow interrupts disabled
05 TSTOP- Timer Stop %1 Timer counter stopped
04 TRST - Timer Reset %0 No effect
03 Not implemented %0 Always returns zero
02-00 PS - Prescaler Select %000 Internal bus clock /1
Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

Peripheral Modules Commands

If you have selected a device (see Device Option), the RS08FCS Menu will display a list of peripheral modules and the associated commands for the device you have chosen.

26

RS08 P&E Multilink/Cyclone PRO Connection

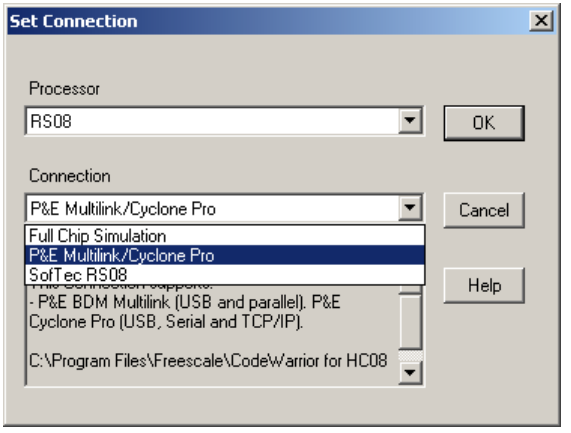
The RS08 P&E Multilink/Cyclone Pro Connection setting permits a connection to Multilink/Cyclone PRO devices. RS08 P&E Multilink/Cyclone Pro connection mode allows the user to debug code, as the firmware is fully resident in the FLASH of the microprocessor. The operation of all modules fully reflects the actual operation of the on-board resources.

Connection Procedure

To select the P&E Multilink/Cyclone Pro as your debugger connection:

1. Choose the P&E Multilink/Cyclone Pro option from the Set Connection dialog box as shown in Figure 26.1 on page 613.
2. Click the OK button.

Figure 26.1 Set Connection Dialog Box



3. Choose the P&E device that you are using from the Interface drop-down menu and click on Refresh. See Figure 26.2 on page 614 and Figure 26.3 on page 615.

Figure 26.2 RS08 Connection Assistant Interface Selection

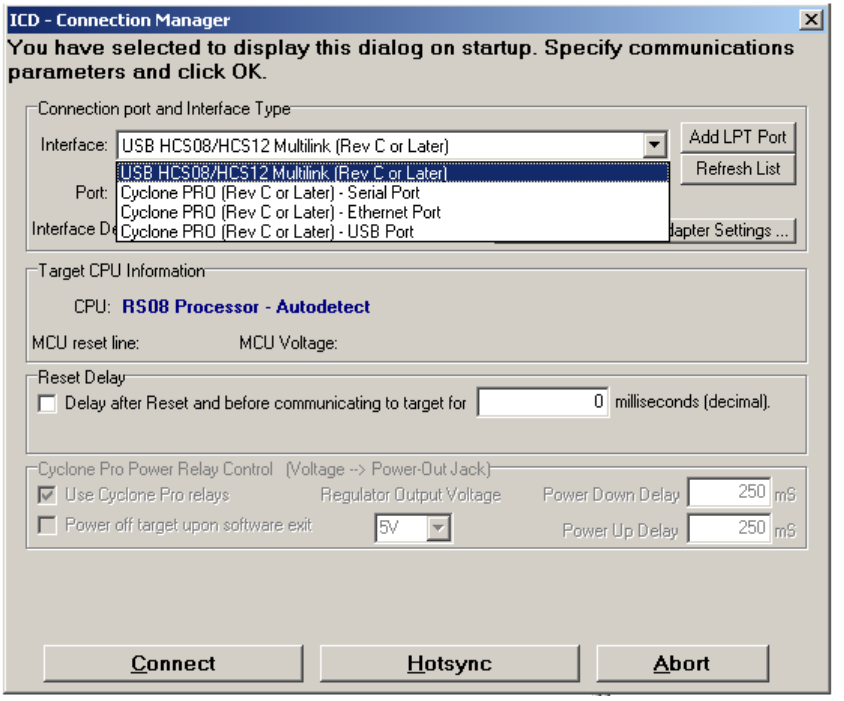
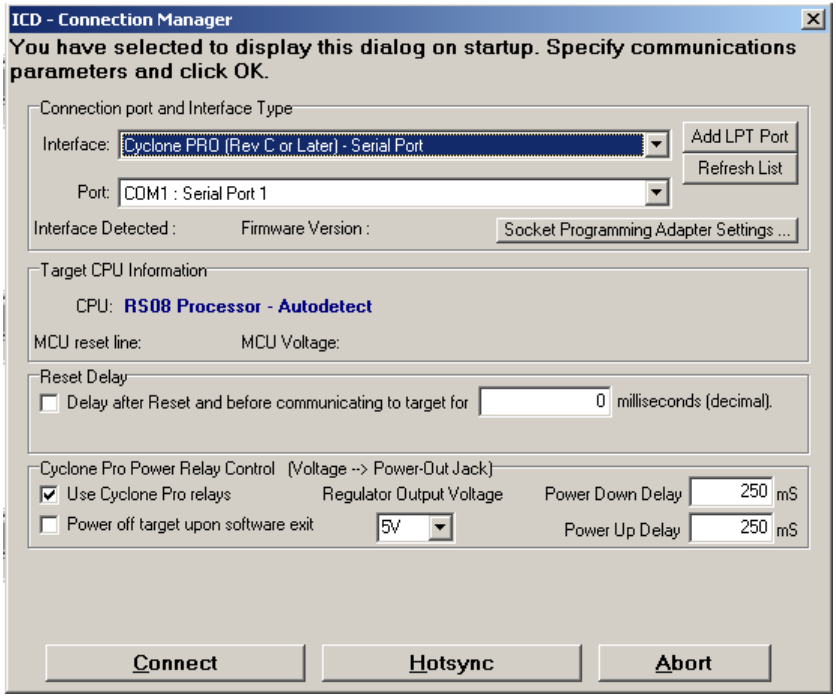


Figure 26.3 RS08 Connection Assistant Interface Selected



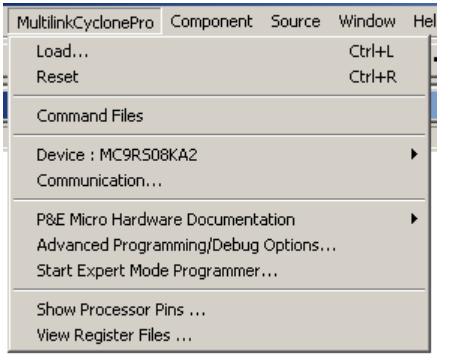
Hotsync Button

The “Hotsync” button in the Connection Assistant (see Figure 26.3 on page 615) allows the user to connect to an already running target.

MultilinkCyclonePro Menu Description

When you have selected P&E Multilink/Cyclone Pro as your connection, the Connection menu's name is changed and other options are added.

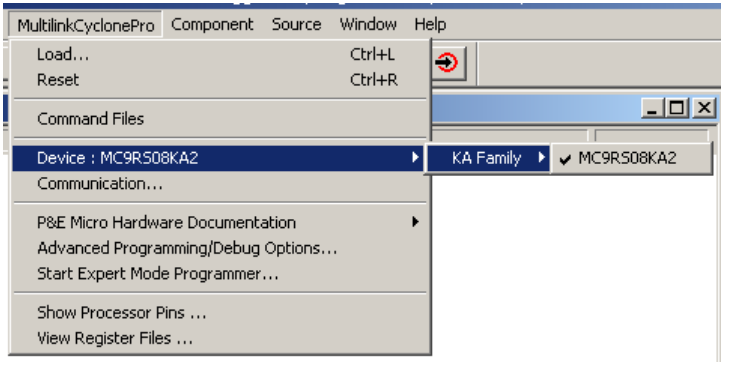
Figure 26.4 Connection (MultilinkCyclonePro) Menu



Device Option

The "Device" option in the MultilinkCyclonePro menu allows the user to select the particular Freescale processor that they wish to use. When choosing Device from the MultilinkCyclonePro menu, extended menus will open which allow you to select the family (e.g. KA Family), and device type (e.g. MC9RS08KA2) of the MCU that you are using.

Figure 26.5 RS08 Device Extended Menu



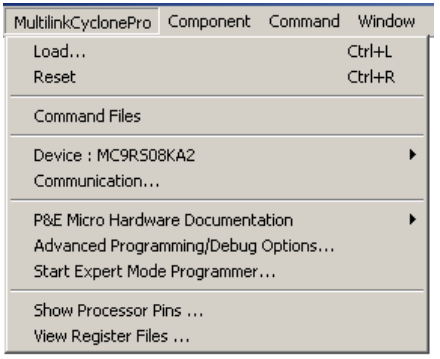
Connect Option

The “Connect” option initiates an attempt to communicate with the device chosen under the Device section of the menu.

Active Mode Menu Options

When the microprocessor is connected, more Connection Menu options become available to the user.

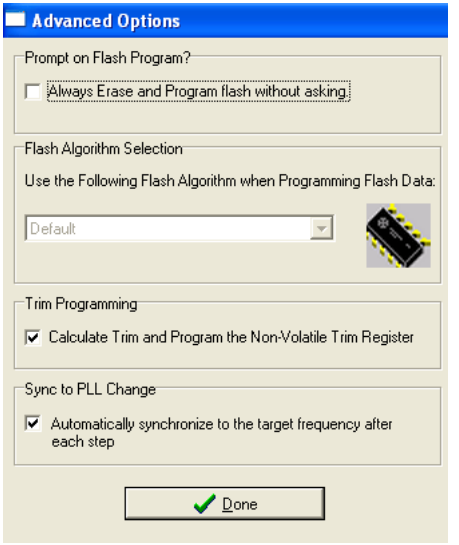
Figure 26.6 Additional Connection Menu Options



Advanced Programming/Debug Options

The Advanced Programming/Debug Options menu option takes you to the Advanced Options dialog box, where you can configure the software settings for the FLASH programming procedure.

Figure 26.7 Advanced Options Dialog Box


Advanced Options
Prompt on Flash Program?
Always Erase and Program flash without asking
Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
Default
Trim Programming
Calculate Trim and Program the Non-Volatile Trim Register
Sync to PLL Change
Automatically synchronize to the target frequency after each step
Done

Prompt on Flash Program Checkbox

Checking “Always Erase and Program flash without asking” in this dialog box lets the software transparently program the microprocessor.

Trim Programming Checkbox

The “Trim Programming” checkbox enables automatic calculation and programming of the trim value in a designated non-volatile memory location.

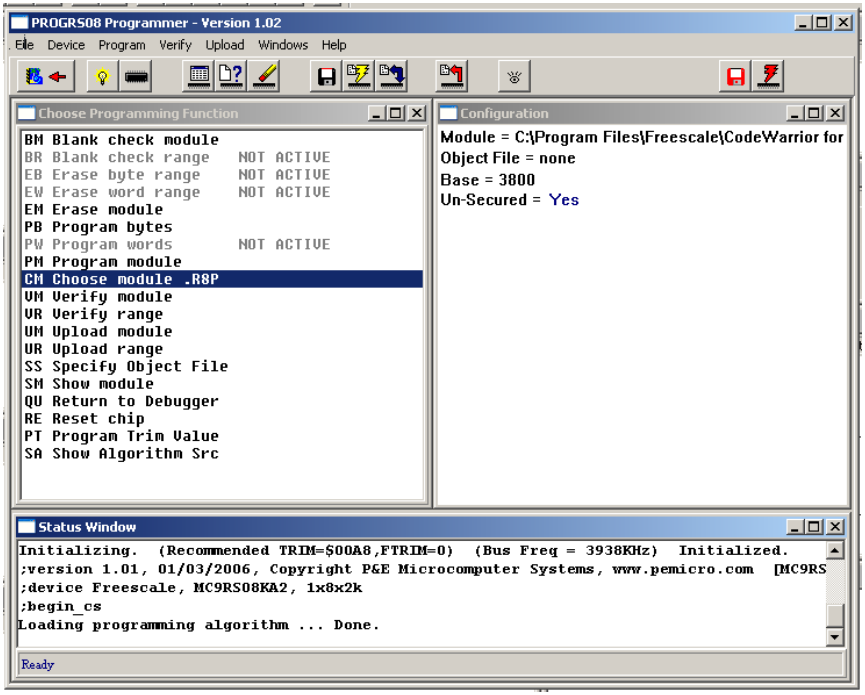
Sync to PLL Change Checkbox

“Sync to PLL Change” is required for the software/hardware connection to synchronize with the microprocessor during the flash erasing/programming procedure.

Start Expert Mode Programmer Option

The Start Expert Mode Programmer option in the Connection Menu grants the user access to P&E’s graphical flash programming utility, PROGRS08. PROGRS08 lets an advanced user control the step-by-step execution of the flash erase/programming procedure. See Figure 26.8 on page 619. More information on how to use the PROGRS08 can be found on the P&E Microcomputer Systems website at www.pemicro.com.

Figure 26.8 PROGRS08 Programmer Window

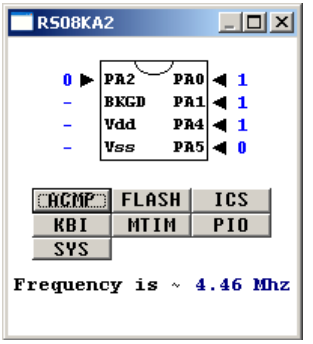

PROGRS08 Programmer - Version 1.02
File Device Program Verify Upload Windows Help
Choose Programming Function
BM Blank check module
BR Blank check range NOT ACTIVE
EB Erase byte range NOT ACTIVE
EW Erase word range NOT ACTIVE
EM Erase module
PB Program bytes
PW Program words NOT ACTIVE
PM Program module
CM Choose module .R8P
VM Verify module
VR Verify range
UM Upload module
UR Upload range
SS Specify Object File
SM Show module
QU Return to Debugger
RE Reset chip
PT Program Trim Value
SA Show Algorithm Src
Configuration
Module = C:\Program Files\Freescale\CodeWarrior for
Object File = none
Base = 3800
Un-Secured = Yes
Status Window
Initializing. (Recommended TRIM=$00A8,FTRIM=0) (Bus Freq = 3938KHz) Initialized.
;version 1.01, 01/03/2006, Copyright P&E Microcomputer Systems, www.pemicro.com [MC9RS
;device Freescale, MC9RS08KA2, 1x8x2k
;begin_cs
Loading programming algorithm ... Done.
Ready

Show Processor Pins

The Processor Pins Dialog is a convenient tool for monitoring the current state of the processor pins, as well as the peripheral with which a pin may be associated. The processor is represented, in graphical form, in the middle of the dialog. Each “pin” is labeled with its current function. For example, Pin 2 will be the BKGD pin upon reset. However, with the appropriate write to the SOPT register, the user can observe that Pin 2 will change to “PTA3” in the Processor Pins Dialog.

In the dialog, each processor pin (with the exception of the power pins, Vdd and GND) has a corresponding arrow. If the arrow points towards the processor, this indicates that the pin is configured as an input. Conversely, if the arrow is pointing away from the processor, this indicates that the pin is configured as an output.

Figure 26.9 Processor Pins Window



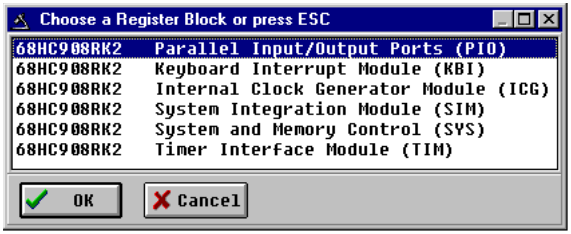
Below the graphical processor representation are several buttons, each corresponding to a processor peripheral. Pressing the button of a given peripheral will bring up the appropriate register files (see below), allowing for easy and informative manipulation of all status, control, and data bits associated with a peripheral.

At the bottom of the dialog is a reading of the current processor speed. This is measured on the device itself, and is a real-time indication of the current bus speed of the processor.

View Register Files Option

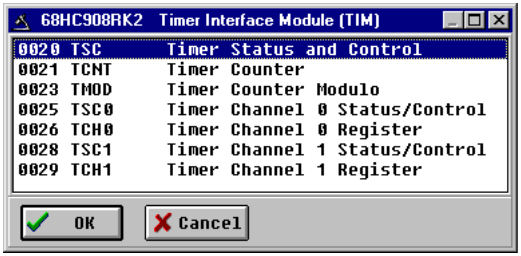
The “View Register Files...” Connection menu selection also gives the user the option of running the register file viewer/editor. If register files are available for the device that you have chosen, the “Choose a Register Block” window (see Figure 26.10 on page 620) will be opened. You may also open it by entering the R command in the Command Window command line.

Figure 26.10 Choose A Register Block Window


Choose a Register Block or press ESC
68HC908RK2 Parallel Input/Output Ports (PIO)
68HC908RK2 Keyboard Interrupt Module (KBI)
68HC908RK2 Internal Clock Generator Module (ICG)
68HC908RK2 System Integration Module (SIM)
68HC908RK2 System and Memory Control (SYS)
68HC908RK2 Timer Interface Module (TIM)
OK
Cancel

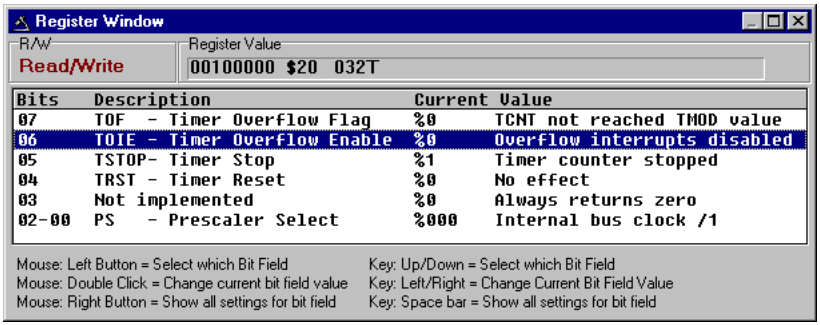
If register files have been installed on the host computer, selecting a block brings up the Register Block register listing (see Figure 26.11 on page 621), which shows a list of the associated registers, their addresses, and their descriptions. This begins interactive setup of system registers such as I/O, timer, and COP watchdog.

Figure 26.11 Register Block Register Listing


68HC908RK2 Timer Interface Module (TIM)
0020 TSC Timer Status and Control
0021 TCNT Timer Counter
0023 TMOD Timer Counter Modulo
0025 TSC0 Timer Channel 0 Status/Control
0026 TCH0 Timer Channel 0 Register
0028 TSC1 Timer Channel 1 Status/Control
0029 TCH1 Timer Channel 1 Register
OK
Cancel

Selecting a file brings up the Register Window (see Figure 26.12 on page 621), which displays the values and significance for each bit in the register. The registers can be viewed and their values modified, and the values can be stored back into debugger memory.

Figure 26.12 Register Window


Register Window
R/W
Read/Write
Register Value
00100000 $20 032T
Bits Description Current Value
07 TOF - Timer Overflow Flag %0 TCNT not reached TMOD value
06 TOIE - Timer Overflow Enable %0 Overflow interrupts disabled
05 TSTOP- Timer Stop %1 Timer counter stopped
04 TRST - Timer Reset %0 No effect
03 Not implemented %0 Always returns zero
02-00 PS - Prescaler Select %000 Internal bus clock /1
Mouse: Left Button = Select which Bit Field
Mouse: Double Click = Change current bit field value
Mouse: Right Button = Show all settings for bit field
Key: Up/Down = Select which Bit Field
Key: Left/Right = Change Current Bit Field Value
Key: Space bar = Show all settings for bit field

27

SofTec RS08 Connection

This section guides you through the first steps toward debugging with CodeWarrior and the *SofTec RS08* connection.

SofTec RS08 Technical Considerations

The 8/16 bit debugger (and then the CodeWarrior IDE) might be connected to RS08 hardware using the SofTec RS08.

When the debugger runs the **SofTec RS08** connection, it can communicate and debug **RS08** core based hardware connected through the SofTec in-circuit debugger/programmer units, such as:

SofTec Microsystems HCS08 ISP Debuggers/Programmers (inDART Series) and Starter Kits (PK and newer Series).

Please refer to the "***inDART®-HCS08 In-Circuit Debugger/Programmer for Motorola HCS08 Family FLASH Devices User's Manual***" from SofTec for communication hardware requirements and SofTec product installation.

CodeWarrior and SofTec RS08 Connection

There are two seperate paths that may be followed to take the first steps toward debugging with Codewarrior and the SofTec inDART-RS08 connection. The differences between the two paths hinge on the starting point for the steps:

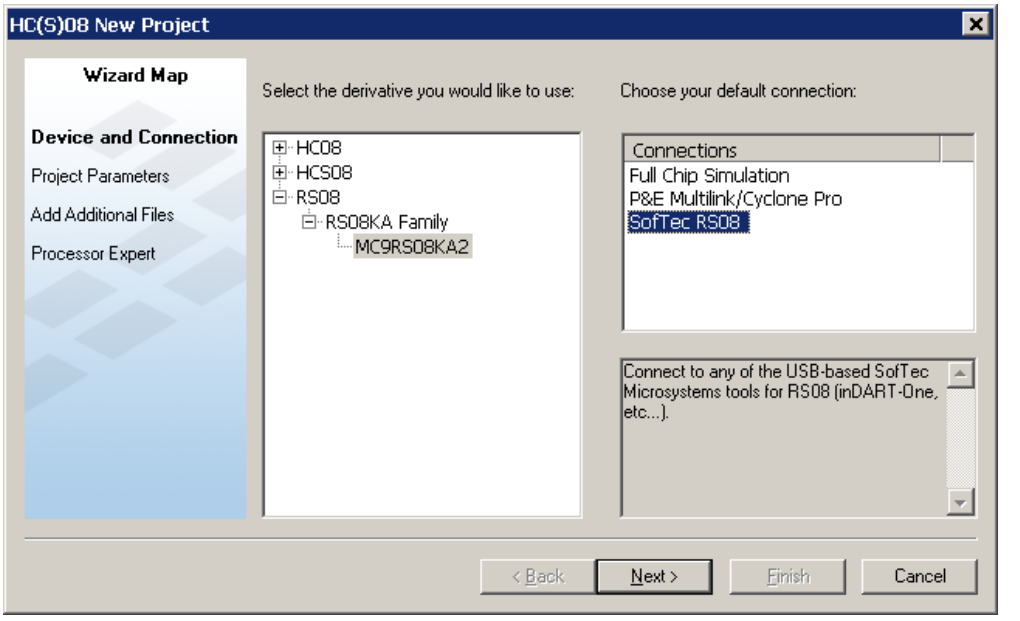
- Using the Stationary Wizard at the start of the project
- From within an existing project

First Steps Using the Stationery Wizard

To take the first steps toward debugging with CodeWarrior and theSofTec inDART-RS08 using the stationery Wizard:

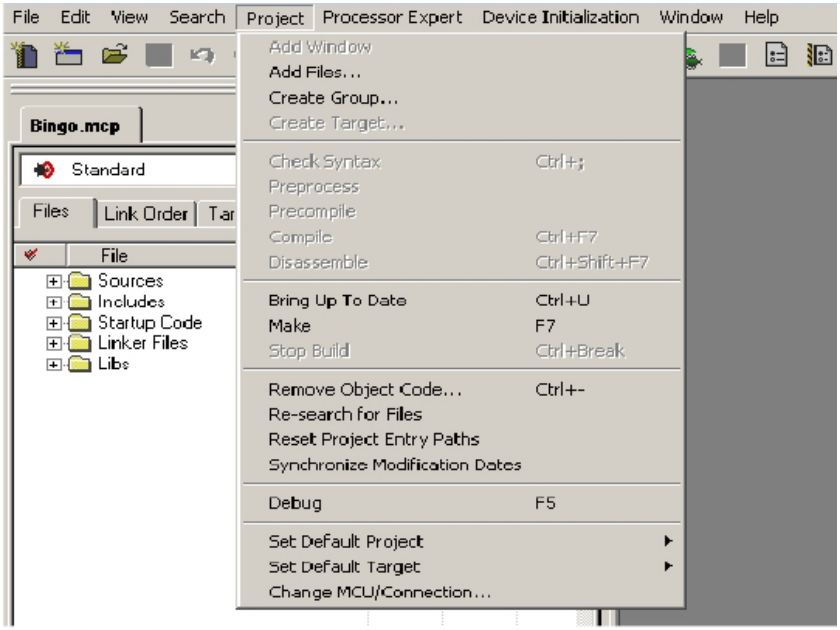
1. Run the *CodeWarrior IDE* with the shortcut created in the program group.
2. In the *HC(S)08 New Project Wizard*, follow the path to create a new project, naming the project.
3. Click the Next button to open the New Project window.
4. In the HC(S)08 New Project window, choose RS08 Family chip you are working with from the list in the Derivative list box in the left of the window.

Figure 27.1 Wizard Connection Selection



5. From the Default Connection list box, choose the connection **SofTec RS08** to create a new project from this stationery.
6. Click the Finish button - IDE opens as shown in Figure 27.2 on page 625.

Figure 27.2 IDE Main Window - Project Menu



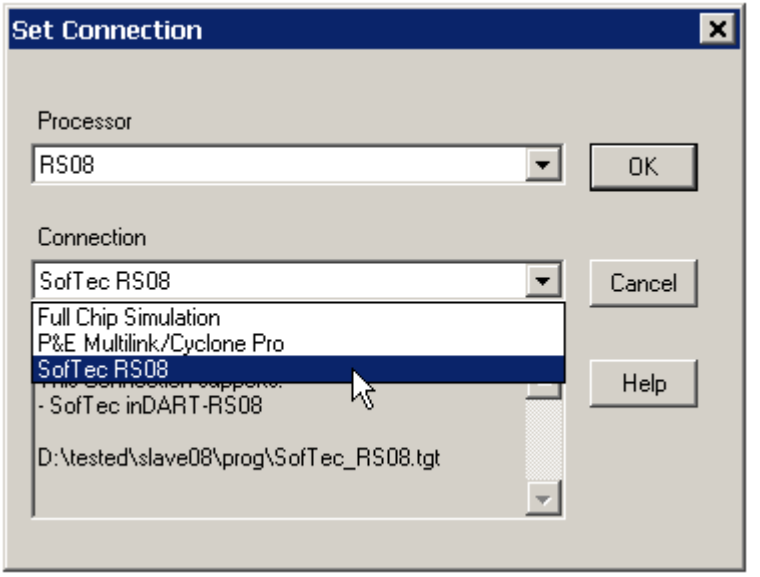
7. Choose the menu option **Project > Make.**
8. Choose the menu option **Project > Debug** to start the debugger.
9. Start debugging.

First Steps From Within an Existing Project

To take the first steps toward debugging with CodeWarrior and setting the SofTec RS08 connection from within an existing debugging project:

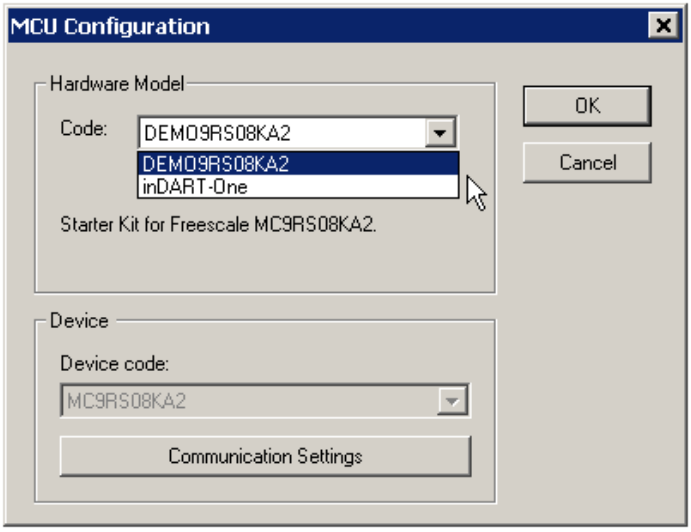
1. Run the *CodeWarrior IDE* with the shortcut created in the program group.
2. Open the project.
3. Choose the menu **Project > Debug** to start the debugger.
4. Choose in the debugger menu **Component > Set Connection..** to select another target interface in the Set Connection dialog box.
5. Select ***RS08*** as the Processor then ***SofTec RS08*** as the connection.

Figure 27.3 Set Connection Dialog Box - SofTec RS08 Selection



6. Press the OK button - MCU Configuration dialog box opens.
7. In the MCU Configuration dialog box, choose the correct target processor.

Figure 27.4 MCU Configuration Dialog Box

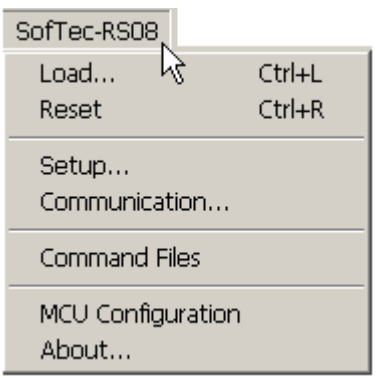


8. Press the OK button to start debugging.

SofTec RS08 Menu Options

Once the ***SofTec RS08*** connection is set, the connection menu entry in the debugger main toolbar is "***SofTec-RS08***".

Figure 27.5 SofTec-RS08 Menu Options



MCU Configuration Option

Select **SofTec-RS08 > MCU Configuration** to display the MCU Configuration Dialog Box.

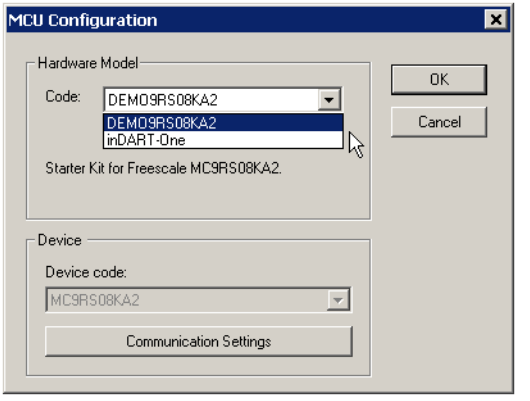
About Option

Select the **SofTec-RS08 > About...** *option* to display the About Dialog Box.

MCU Configuration Dialog Box

The Hardware Model drop down list can be expanded to select another type of BDC debug interface than the SofTec inDART-RS08. The *HW Code* drop down list can be expanded to select another RS08 derivative. Note that at the time of this document release, only the SofTec inDART-RS08 is available.

Figure 27.6 MCU Configuration Dialog Box



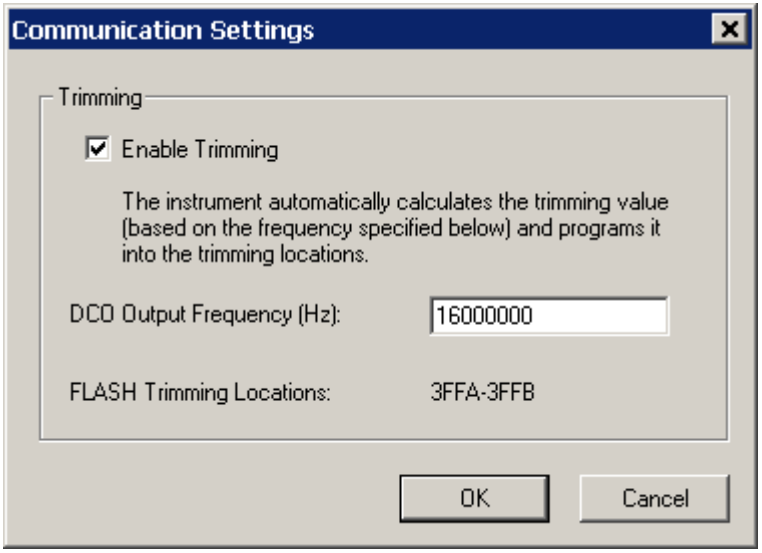
Pressing the *Communication Settings* button in this window opens the Communication Settings Dialog Box.

Communication Settings Dialog Box

The communication dialog prives chip trimming. The "*Enable Trimming*" checkbox can be checked to enable the trimming data calculation according to the requested DCO value that can be specified in the "*DCO Output Frequency (Hz)*" edit box. The trimming data is then programmed automatically at application loading time at specified locations (*Flash Trimming locations*).

Refer to the ***"inDART®-HCS08 In-Circuit Debugger/Programmer for Motorola HCS08 Family FLASH Devices User's Manual"*** from SofTec for further details.

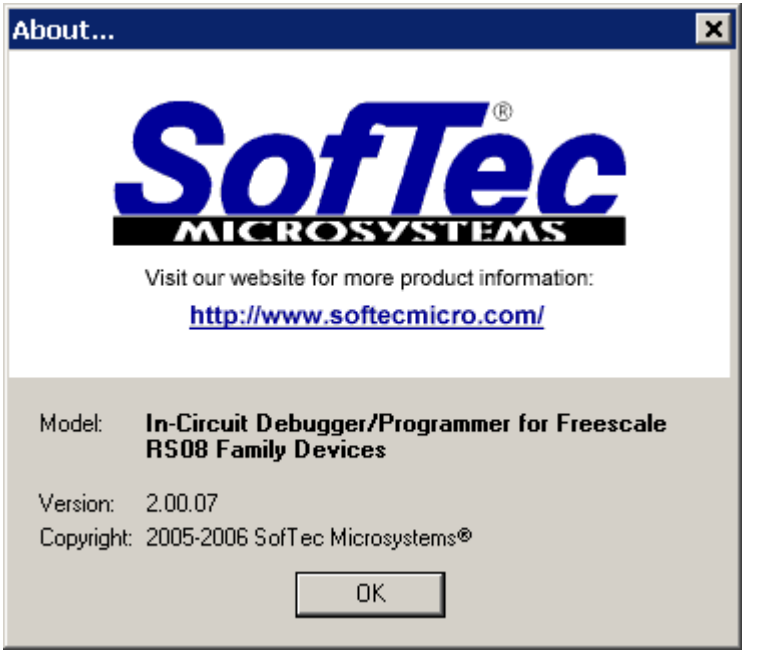
Figure 27.7 Communication Settings Dialog box



About Dialog Box

This dialog box belongs to the SofTec GDI DLL and provides information about the ***SofTec_BDC08.dll*** release and version.

Figure 27.8 About Dialog Box



Book V - Commands

Book V Contents

Each section of the Debugger manual includes information to help you become more familiar with the Debugger, to use all its functions and help you understand how to use the environment. This book, the Debugger Commands, defines the HC08 and HC(S)08 Commands, both those commands used by the debugger engine and those specific to individual debugger connections.

This book is divided into the following chapters:

Book 5: Commands

- Debugger Engine Commands on page 633
- Debugger Connection-specific Commands on page 745

28

Debugger Engine Commands

Commands Overview

The debugger supports scripting with the use of commands and command files. When you script the debugger, you can automate repetitive, time-consuming, or complex tasks.

You do not need to use or have knowledge of commands to run the Simulator/Debugger. However these commands are useful for editing debugger command files, for example, after a recording session, to generate your own command files, or to set up your applications and targets, etc.

This section provides a detailed list of all Simulator/Debugger commands. All command names and component names are case insensitive. The command EBNF syntax is:

```
component [:component number] < ] command
```

where **component** is the name of the component that you can read in each component window title. For example: Data, Register, Source, Assembly, etc. **Component number** is the number of the component. This number does not exist in the component window title if only one component of this type is open. For example, you will read **Register** or **Memory.** If you open a second Memory component window, the initial one will be renamed **Memory:1** and the new one will be called **Memory:2**. A number is automatically associated with a component if there are several components of the same type displayed.

Command Example:

```
in>Memory:2 < SMEM 0x8000,8
```

'<' redirects a command to a specific component (in this example: **Memory:2**). Some commands are valid for several or all components and if the command is not redirected to a specific component, all components will be affected. Also, a mismatch could occur due to the fact that a command's parameters could differ for different components.

Command Syntax

To display the syntax of a command, type the command followed by a question mark.

Syntax Example:

```
in>printf?
PRINTF (<format>, <expression>, <expression>, ...)
```

Available Command Lists

Commands described on the following pages are sorted into 5 groups, according to their specific actions or targets. However, these groups have no relevance in the use of these commands. A list of all commands in their respective group is given below:

Kernel Commands

Kernel commands are commands that can be used to build command programs. They can only be used in a debugger command file, since the Command Line component can only accept one command at a time. It is possible to build powerful programs by combining Kernel commands with Base commands, Common commands and Component specific commands. Table 28.1 on page 634 contains all available Kernel commands.

Table 28.1 List of Kernel Commands

Command, Syntax	Short Description
A on page 644	Affects a **value**
AT on page 655	Sets a time delay for command execution
CALL on page 661 fileName**[;C][;NL]**	Executes a command file
DEFINE on page 672 symbol [=] expression	Defines a user symbol
ELSE on page 677	Other operation associated with **IF** command
ELSEIF on page 677 condition	Other conditional operation associated with **IF** command
ENDFOCUS on page 678	Resets the current focus (refer to **FOCUS** command)

Table 28.1 List of Kernel Commands (*continued*)

Command, Syntax	Short Description
ENDFOR on page 679	Exits a **FOR** loop
ENDIF on page 679	Exits an **IF** condition
ENDWHILE on page 680	Exits a **WHILE** loop
FOCUS on page 685 component	Sets the focus on a specified component
FOR on page 687 [variable =]range [“,” step]	**FOR** loop instruction
FPRINTF on page 688 (fileName,format,parameters**)**	FPRINTF instruction
GOTO on page 691 label	Unconditional branch to a label in a command file
GOTOIF on page 692 condition Label	Conditional branch to a label in a command file
IF on page 694 condition	Conditional execution
PAUSETEST on page 712	Displays a modal message box
PRINTF on page 713 (“Text:,” value]**)**	PRINT instruction
REPEAT on page 715	REPEAT loop instruction
RETURN on page 717	Returns from a CALL command
TESTBOX on page 734	Displays a message box with a string
UNDEF on page 735 symbol | *	Undefines a userdefined symbol
UNTIL on page 738 condition	Condition of a REPEAT loop
WAIT on page 740 [time] [;s]	Command file execution pause
WHILE on page 742 condition	WHILE loop instruction

Base Commands

Base commands are used to monitor the Simulator/Debugger target execution. Target input/output files, target execution control, direct memory editing, breakpoint management and CPU register setup are handled by these commands. Base commands can

be executed independent of components that are open. Table 28.2 on page 636 contains all available Base commands.

Table 28.2 List of Base Commands

Command, Syntax	Short Description
BC on page 656 address|*	Deletes a breakpoint (breakpoint clear)
BS on page 659 address|function [P|T[state]]	Sets a breakpoint (breakpoint set)
CD on page 662 [path]	Changes the current working directory
CR on page 667 [fileName][;A]	Opens a record file (command records)
DASM on page 668 **[address|range][;OBJ]**	Disassembles
DB on page 669 [address|**range]**	Displays memory bytes
DL on page 674 [address|**range]**	Displays memory bytes as longwords
DW on page 675 [address|**range]**	Displays memory bytes as words
G on page 689 [address]	Starts execution of the application currently loaded
GO on page 690 [address]	Starts execution of the application currently loaded
LF on page 696 [fileName][;A]	Opens a log file
LOG on page 699 type [=] state {[,] type [=] state}	Enables or disables logging of a specified information type
MEM on page 704	Displays the memory map
MS on page 705 range list	Sets memory bytes
NOCR on page 708	Closes the record file
NOLF on page 708	Closes the log file
P on page 711 [address]	Single assembly steps into program
RESTART on page 716	Restarts the loaded application
RD on page 714 [list|*]	Displays the content of registers

Table 28.2 List of Base Commands (*continued*)

Command, Syntax	Short Description
RS on page 718 register[=]value{,register[=]value}	Sets a register
S on page 719	Stops execution of the loaded application
STEPINTO on page 729	Steps to the next source instruction of the loaded application
STEPOUT on page 730	Executes program out of a function call
STEPOVER on page 731	Steps over the next source instruction of the loaded application
STOP on page 732	Stops execution of the loaded application
SAVEBP on page 721 on|off	Saves breakpoints
T on page 733 [address][,count]	Traces program instructions at the specified address
WB on page 741 range list	Writes bytes
WL on page 743 range lis	Writes longwords
WW on page 743 range list	Writes words

Environment Commands

Simulator/Debugger environment commands are used to monitor the debugger environment, specific component window layouts and framework applications and targets. Table 28.3 on page 637 contains all available Environment commands.

Table 28.3 List of Environment Commands

Command, Syntax	Short Description
ACTIVATE on page 644 component	Activates a component window
AUTOSIZE on page 655 on|off	Autosizes windows in the main window layout
BCKCOLOR on page 657 color	Sets the background color
CLOSE on page 665 component | *	Closes a component

Table 28.3 List of Environment Commands (*continued*)

Command, Syntax	Short Description
DDEPROTOCOL on page 671 ON|OFF|SHOW|HIDE|STATUS	Configures the Debugger/Simulator DDE protocol
FONT on page 686 'fontName' [size][color]	Sets text font
LOAD on page 697 applicationName	Loads a framework application (code and debug information)
LOADCODE on page 698 applicationName	Loads the code of a framework application
LOADSYMBOLS on page 698 applicationName	Loads debugging information of a framework application
OPEN on page 709 component [[x y width height][;][i|max]]	Opens a Windows component
SET on page 722 targetName	Sets a new target
SLAY on page 723 fileName	Saves the general window layout

Component Commands

Component common commands are used to monitor component behaviors. They are common to more than one component and for better usage, they should be redirected (as explained in the introduction to this chapter. Table 28.4 on page 638 contains all available Component commands.

Table 28.4 List of Component Commands

Command, Syntax	Short Description
CMDFILE on page 666	Specify a command file state and full name
EXIT on page 681	Terminates the application
HELP on page 693	Displays a list of available commands
RESET on page 716	Resets statistics
SMEM on page 724 range	Shows a memory range
SMOD on page 725 module	Shows module information in the destination component

Table 28.4 List of Component Commands

Command, Syntax	Short Description
SPC on page 726 address	Shows the specified address in a component window
SPROC on page 727 level	Shows information associated with the specified procedure
VER on page 739	Displays version number of components and engine

Component Specific Commands

Component specific commands are associated with specific components. Table 28.5 on page 639 contains all available Component Specific commands.

Table 28.5 List of Component Specific Commands

Command, Syntax	Short Description
ADDXPR on page 645 “expression”	Adds a new expression in the data component
ATTRIBUTES on page 645 list	Sets up the display inside a component window
BASE on page 656 code | module	Sets the Profiler base
BD on page 658	Displays a list of all breakpoints
CF on page 663 fileName [;C][;NL]	Executes a command file
CLOCK on page 665 frequency	Sets the clock speed
COPYMEM on page 666 <Source addr range> dest-addr	Copies memory
CYCLE on page 667 on|off	Switches cycles and milliseconds in SofTrace component.
DETAILS on page 673 assembly|source	Sets split view
DUMP on page 675	Displays data component content

Table 28.5 List of Component Specific Commands (*continued*)

Command, Syntax	Short Description
E on page 676 expression [;O|D|X|C|B]	Evaluates a given expression
EXECUTE on page 680 fileName	Executes a stimulation file
FILL on page 681 range value	Fills a memory range with a value
FILTER on page 682 Options [<range>]	Selects the output file filter options
FIND on page 683 “string” [;B] [;MC] [;WW]	Finds and highlights a pattern
FINDPROC ON PAGE 684 ProcedureName	Opens a procedure file
FOLD on page 686 [*]	Folds a source block
FRAMES on page 688 number	Sets the maximum number of frames
GRAPHICS on page 693 on|off	Switches graphic bars on/off
INSPECTOROUTPUT on page 695 [name {subname}]	Prints content of Inspector to Command window
INSPECTORUPDATE on page 695	Updates content of Inspector
LS on page 703 [symbol | *][;C|S]	Displays the list of symbols
NB on page 706 [base]	Sets the base of arithmetic operations
OUTPUT on page 710 fileName	Redirects the coverage component results
PTRARRAY on page 713 on|off	Switches on /off the pointer as array display
RECORD on page 715 on|off	Switches on/off the frame recorder
SLINE on page 723 linenumber	Shows the desired line number
SAVE on page 720 range fileName [offset][;A]	Saves a memory block in S-Record format
SETCOLORS on page 722 ("Name") (Background) (Cursor) (Grid) (Line) (Text)	Changes the colors attributes of the "Name" channel from the Monitor component

Table 28.5 List of Component Specific Commands (*continued*)

Command, Syntax	Short Description
SREC on page 728 fileName [offset]	Loads a memory block in S-Record format
TUPDATE on page 734 on|off	Switches on/off time update for statistics
UNFOLD on page 737 [*]	Unfolds a source block
UPDATERATE on page 738 rate	Sets the data and memory update mode
ZOOM on page 744 address in|out	Zooms in/out a variable

Command Syntax Terms

address

A number matching a memory address. This number must be in the ANSI format (i.e. **$** or **0x** for hexadecimal value, **0** for octal, etc.).

Example: 255, 0377, 0xFF, $FF

NOTE **`address`** can also be an "expression" if "constant address" is not specially mentioned in the command description. An "expression" can be: Global variables of application, I/O registers defined in `DEFAULT.REG`, definitions in the command line, numerical constants.

Example: **DEFINE IO_PORT = 0x210**

WB IO_PORT 0xFF

range

A composition of 2 addresses to define a range of memory addresses. Syntax is shown below:

address..**address**

or

address, **size**

where **size** is an ANSI format numerical constant.

Example:

0x2F00..0x2FFF

Refers to the memory range starting at **0x2F00** and ending at **0x2FFF** (256 bytes).

Example:

0x2F00,256

Refers to the memory range starting at **0x2F00**, which is 256 bytes wide. Both previous examples are equivalent.

fileName

A DOS file name and path that identifies a file and its location. The command interpreter does not assume any file name extension. Use backslash (\) or slash (/) as a directory delimiter.

The parser is case insensitive. If no path is specified, it looks for (or edits) the file in the current project directory, i.e. when no path is specified, the default directory is the project directory.

Example:

d:/demo/myfile.txt

Example:

layout.hwl

Example:

d:/work/project.hwc

component

The name of a debugger component. A list of all debugger components is given by choosing **Component>Open...** The parser is case insensitive.

Example:

Memory

Example:

SoUrCe

Module Names

Correct module names are displayed in the Module component window. Make sure that the module name of a command that you implement is correct:

If the `.abs` is in **HIWARE** format, some debug information is in the object file (`.o`), and module names have a `.o` extension (e.g., fibo`.o`).

In **ELF** format, module name extensions are `.c`, `.cpp` or `.dbg` (`.dbg` for program sources in assembler) (e.g., `fibo.c`), since all debugging information is contained in the `.abs` file and object files are not used.

Debugger Commands

The commands available when you use the Simulator/Debugger are defined on the following pages.

A

The **A** command assigns an expression to an existing variable. The quoted expression must be used for string and enum expressions.

Usage

A variable = value or A variable = "value"

Components

Debugger engine.

Example:

```
in>a counter=8
```

The variable **counter** is now equal to **8**.

```
in>A day1 = "monday_8U"    (Monday_8U is defined in an Enum)
```

The variable **day1** is now equal to **monday_8U**.

```
in>A value = "3.3"
```

The variable **value** is now equal to **3.3**

ACTIVATE

ACTIVATE activates a component window as if you clicked on its title bar. The window is displayed in the foreground and its title bar is highlighted. If the window is iconized, its title bar is activated and displayed in the foreground.

Usage

ACTIVATE component

Components

Debugger engine.

Example:

```
in>ACTIVATE Memory
```

Activates the Memory Component and brings the window to the foreground.

ADDXPR

The **ADDXPR** command adds a new expression in the data component.

Usage

ADDXPR “expression”

Where the parameter expression is an expression to be added and evaluated in the data component.

Components

Data component.

Example:

```
in>ADDXPR “counter + 10”
```

The expression “counter +10” is added in the data component.

ATTRIBUTES

This command is effective for various components as described in the next sections.

In the Command Component

The ATTRIBUTES command allows you to set the display and state options of the Command component window. The CACHESIZE command sets the cache size in lines for the Command Line window: The cache size value is between 10 and 1000000.

NOTE Usually this command is not specified interactively by the user. However this command can be written in a command file or a layout (`".HWL"`) file to save and reload component window layouts. An interactive equivalent operation is typically possible, using Simulator/Debugger menus and operations, drag and drops, etc., as described in the following sections in “Equivalent Operations”.

Usage

ATTRIBUTES list

where list=command{,command})

command=CACHESIZE value

Example:

```
command < ATTRIBUTES 2000
```

In the Procedure Component

The **ATTRIBUTES** command allows you to set the display and state options of the Procedure component window. The **VALUES** and **TYPES** commands display or hide the Values or Types of the parameters.

Usage

ATTRIBUTES list

where **list=command{,command}**)

command=**VALUES (ON|OFF)| TYPES (ON|OFF)**

Example:

```
Procedure < ATTRIBUTES VALUES ON,TYPES ON
```

In the Assembly Component

The **ATTRIBUTES** command allows you to set the display and state options for the Assembly component window. The **ADR** command displays or hides the address of a disassembled instruction. **ON | OFF** is used to switch the address on or off. **SMEM** (show memory range) and **SPC** (show PC address) scroll the Assembly component to the corresponding address or range code location and select/highlight the corresponding assembler lines or range of code. The **CODE** command displays or hides the machine code of the disassembled instruction. **ON | OFF** is used to switch on or off the machine code. The **ABSADR** command shows or hides the absolute address of a disassembled instruction like ‘branch to’. **ON | OFF** is used to switch on or off the absolute address. The **TOPPC** command scrolls the Assembly component in order to display the code location given as an argument on the first line of Assembly component window. The **SYMB** command displays or hides the symbolic names of objects. **ON | OFF** is used to switch the symbolic display on or off.

Usage

ATTRIBUTES list

where **list=command{,command}**

command= **ADR (ON|OFF) | SMEM range | SPC address | CODE(ON|OFF) | ABSADR (ON|OFF) | TOPPC address | SYMB (ON|OFF)**

NOTE Also refer to SMEM on page 724 and SPC on page 726 command descriptions for more detail about these commands. The **SPC** command is similar to the **TOPPC** command but also highlights the code and does not scroll to the top of the component window.

Equivalent Operations

ATTRIBUTES ADR ~ Select menu **`Assembly>Display Adr`**

ATTRIBUTES SMEM ~ Select a range in Memory component window and drag it to the Assembly component window.

ATTRIBUTES SPC ~ Drag a register to the Assembly component window.

ATTRIBUTES CODE ~ Select menu **Assembly>Display Code**

ATTRIBUTES SYMB ~ Select menu **Assembly>Display `Symbolic`**

Example:

```
Assembly < ATTRIBUTES ADR ON,SYMB ON,CODE ON, SMEM 0x800,16
```

Addresses, hexadecimal codes, and symbolic names are displayed in the Assembly component window, and assembly instructions at addresses 0x800,16 are highlighted.

In the Register Component

The **ATTRIBUTES** command allows you to set the display and state options of the Register component window.

The **FORMAT** command sets the display format of register values.

The **VSCROLLPOS** command sets the current absolute position of the vertical scroll box (the **vposition** value is in **lines**: each register and bitfield have the same height, which is the height of a **line**). **vposition** is the absolute vertical scroll position. The value **0** represents the first position at the top.

The **HSCROLLPOS** command sets the position of the horizontal scroll box (the **hposition** value is in **columns**: a **column** is about a tenth of the greatest register or bitfield width). **hposition** is the absolute horizontal scroll position. The value **0** represents the first position on the left.

The parameters **vposition** and **hposition** can be constant expressions or symbols defined with the **DEFINE** command.

The **COMPLEMENT** command sets the display complement format of register values:
one sets the first complement (each bit is reversed),
none unselects the first complement.

An error message is displayed if:

- the parameter is a negative value
- the scroll box is not visible

If the given scroll position is bigger than the maximum scroll position, the current absolute position of the scroll box is set to the maximum scroll position.

Equivalent Operations

ATTRIBUTES FORMAT ~ Select menu **Register>Options**

ATTRIBUTES VSCROLLPOS ~ Scroll vertically in the Register component window.

ATTRIBUTES HSCROLLPOS ~ Scroll horizontally in the Register component window.

ATTRIBUTES COMPLEMENT ~ Select menu **Register>Options**

Usage

ATTRIBUTES list

where **list=command{,command}**)

command= **FORMAT (hex|bin|dec|udec|oct)** | **VSCROLLPOS vposition** | **HSCROLLPOS hposition** | **COMPLEMENT(none|one)**

Where **vposition=expression** and **hposition=expression**

Example:

```
in>Register < ATTRIBUTES FORMAT BIN
```

Contents of registers are displayed in binary format in the Register component window.

```
in>Register < ATTRIBUTES VSCROLLPOS 3
```

Scrolls 3 positions down. The third line of registers is displayed on the top of the register component.

```
in>Register < ATTRIBUTES VSCROLLPOS 0
```

Returns to the default display. The first line of registers is displayed on the top of the register component.

```
in>DEFINE vpos = 5
in>Register < ATTRIBUTES HSCROLLPOS vpos
```

Scrolls 5 positions right. The second column of registers is displayed on the left of the register component.

```
in>Register < ATTRIBUTES HSCROLLPOS 0
```

Returns to the default display. The first column of registers is displayed on the left of the register component.

```
in>Register < ATTRIBUTES COMPLEMENT One
```

Sets the first complement display option. All registers are displayed in reverse bit.

In the Source Component

The **ATTRIBUTES** command allows you to set the display and state options of the Source component window. The **SMEM** (show memory range) command and **SPC** (show PC address) command loads the corresponding module's source text, scrolls to the corresponding text range location or text address location and highlights the corresponding statements. The **SMOD** (show module) command loads the corresponding module's source text. If the module is not found, a message is displayed in the Component Windows Object Info Bar on page 35. The **SPROC** (show procedure) command loads the corresponding module's source text, scrolls to the corresponding procedure and highlights the statement, that is in the procedure chain of this procedure. The **numberAssociatedToProcedure** is the level of the procedure in the procedure chain. The **MARKS** command (**ON** or **OFF)** displays or hides the marks.

NOTE Also refer to SMEM on page 724 SPC on page 726, SPROC on page 727 and SMOD on page 725 command descriptions for more detail about these commands.

Equivalent Operations

ATTRIBUTES SPC ~ Drag and drop from Register component to Source component.

ATTRIBUTES SMEM ~ Drag and drop from Memory component to Source component.

ATTRIBUTES SMOD ~ Drag and drop from Module component to Source component.

ATTRIBUTES SPROC ~ Drag and drop from Procedure component to Source component.

ATTRIBUTES MARKS ~ Select menu **Source>Marks** .

Usage

ATTRIBUTES list

where **list=command{,command}**

command= **SPC address | SMEM range** | **SMOD module** (without extension) **| SPROC numberAssociatedToProcedure | MARKS (ON|OFF)**

Example:

```
in>Source < ATTRIBUTES MARKS ON
```

Marks are visible in the Source component window.

In the Data Component

The **ATTRIBUTES** command allows you to set the display and state options of the Data component window. The **FORMAT** command selects the format for the list of variables. The format is one of the following: binary, octal, hexadecimal, signed decimal, unsigned decimal or symbolic.

Usage

ATTRIBUTES list

where **list=command{,command}**)

command=**FORMAT(bin|oct|hex|signed|unsigned|symb)| SCOPE (global|local|user) | MODE (automatic|periodical| locked|frozen) | SPROC level | SMOD module | UPDATERATE rate | COMPLEMENT(none|one)| NAMEWIDTH width**

The **MODE** command selects the display mode of variables.

- In **Automatic** mode (default), variables are updated when the target is stopped. Variables from the currently executed module or procedure are displayed in the data component. Variables are updated when target is stopped.
- In **Locked** and **Frozen** mode, variables from a specific module are displayed in the data component. The same variables are always displayed in the data component.
- In **Locked** mode, values from variables displayed in the data component are updated when the target is stopped.
- In **Frozen** mode, values from variables displayed in the data component are not updated when the target is stopped.
- In **Periodical** mode, variables are updated at regular time intervals when the target is running. The default update rate is 1 second, but it can be modified by steps of up to 100 ms using the associated dialog box or the **UPDATERATE** command.

The **UPDATERATE** command sets the variables update rate (see also UPDATERATE on page 738 command).

The **SPROC** (show procedure) and **SMOD** (show module) commands display local or global variables of the corresponding procedure or module.

The **SCOPE** command selects and displays global, local or user defined variables.

The **COMPLEMENT** command sets the display complement format of Data values: one sets the first complement (each bit is reversed), none unselects the first complement.

The **NAMEWIDTH** command sets the length of the variable name displayed in the window.

NOTE Refer to SPROC on page 727, UPDATERATE on page 738 and SMOD on page 725 command descriptions for more detail about these commands.

Equivalent Operations

ATTRIBUTES FORMAT ~ Select menu **Data>Format...**

ATTRIBUTES MODE ~ Select menu **Data>Mode...**

ATTRIBUTES SCOPE ~ Select menu **Data>Scope...**

ATTRIBUTES SPROC ~ Drag and drop from Procedure component to Data component.

ATTRIBUTES SMOD ~ Drag and drop from Module component to Data component.

ATTRIBUTES UPDATERATE ~ Select menu **Data>Mode>Periodical** .

ATTRIBUTES COMPLEMENT ~ Select menu **Data>Format...**

ATTRIBUTES NAMEWIDTH ~ Select menu **Data>Options...>Name Width...**

Example:

```
Data:1 < ATTRIBUTES MODE FROZEN
```

In **Data:1** (global variables), variables update is frozen mode. Variables are not refreshed when the application is running.

In the Memory Component

The **ATTRIBUTES** command allows you to set the display and state options of the Memory component window. The **WORD** command selects the word size of the memory dump window. The word size **number** can be **1** (for “byte” format), **2** (for “word” format - 2 bytes) or **4** (for “long” format - 4 bytes). The **ADR** command **ON** or **OFF** displays or hides the address in front of the memory dump lines. The **ASC** command **ON** or **OFF** displays or hides the ASCII dump at the end of the memory dump lines. The **ADDRESS** command scrolls the corresponding memory dump window and displays the corresponding memory address lines (memory **WORD** is not selected). **SPC** (show pc), **SMEM** (show memory) and **SMOD** (show module) commands scroll the Memory component accordingly, to display the code location given as argument, and select the corresponding memory area (**SPC** selects an address, **SMEM** selects a range of memory and **SMOD** selects the module name whom global variable would be located in the window).

The **FORMAT** command selects the format for the list of variables. The format is one of the following: binary, octal, hexadecimal, signed decimal, unsigned decimal or symbolic.

The **COMPLEMENT** command sets the display complement format of memory values: one sets the first complement (each bit is reversed), none unselects the first complement.

The **MODE** command selects the display mode of memory words.

- In **Automatic** mode (default), memory words are updated when the target is stopped. Memory words from the currently executed module or procedure are displayed in the Memory component. Memory words are updated when target is stopped.

- In **Frozen** mode, value from memory words displayed in the Memory component are not updated when the target is stopped.
- In **Periodical** mode, memory words are updated at regular time intervals when the target is running. The default update rate is 1 second, but it can be modified by steps of up to 100 ms using the associated dialog box or **UPDATERATE** command.

The **UPDATERATE** command sets the variables update rate (see also UPDATERATE on page 738 command).

NOTE Also refer to SMEM on page 724, SPC on page 726 and SMOD on page 725 command descriptions for more detail about these commands.

Equivalent Operations

ATTRIBUTES FORMAT ~ Select menu **Memory>Format**

ATTRIBUTES WORD ~ Select menu **Memory>Word Size**

ATTRIBUTES ADR ~ Select menu **Memory>Display>Address**

ATTRIBUTES ASC ~ Select menu **Memory>Display>ASCII**

ATTRIBUTES ADDRESS ~ Select menu **Memory>Address...**

ATTRIBUTES COMPLEMENT ~ Select menu **Memory>Format**

ATTRIBUTES SMEM ~ Drag and drop from Data component (variable) to Memory component.

ATTRIBUTES SMOD ~ Drag and drop from Source component to Memory component.

ATTRIBUTES MODE ~ Select menu **Memory>Mode...**

ATTRIBUTES UPDATERATE ~ Select menu **Memory>Mode>Periodical**

Usage

ATTRIBUTES list

where **list=command{,command}**)

command=**FORMAT(bin|oct|hex|signed|unsigned) | WORD number | ADR (ON|OFF) | ASC (ON|OFF) | ADDRESS address | SPC address | SMEM range | SMOD module | MODE (automatic|periodical| frozen) | UPDATERATE rate | COMENT (NONE|ONE)**

Example:

```
Memory < ATTRIBUTES ASC OFF, ADR OFF
```

ASCII dump and addresses are removed from the Memory component window.

In the Inspector Component

The **ATTRIBUTES** command allows you to set the display and state of the Inspector component window.

Usage

ATTRIBUTES list

where **list=command{,command}**)

command= **COLUMNWIDTH columnname columnfield columnsize |**
EXPAND [name {subname}] deep |
COLLAPSE name {subname}|
SELECT name {subname} |
SPLIT pos |
MAXELEM (ON | OFF) [number] |
FORMAT (Hex|Int)

The **COLUMNWIDTH** command sets the width of one column entry on the right pane of the Inspector Window. The first parameter (columnname) specifies which column. The following column names currently exist:

- Names - simple name list
- Interrupts - interrupt list
- SymbolTableFunction - function in the Symbol Table
- ObjectPoolObject - Object in Object Pool without additional information
- Events - event list
- Components - component list
- SymbolTableVariable - variable or differentiation in the Symbol Table
- ObjectPoolIOBase - Object in Object Pool with additional information
- SymbolTableModules - non IOBase derived Object in the Object Pool

The column field is the name of the specific field, which is also displayed in the Inspector Window.

The following commands set the width of the function names to 100:

```
inspect < ATTRIBUTES COLUMNWIDTH SymbolTableModules Name 100
```

NOTE Due to the “inspect <“ redirection, only the Inspector handles this command.

The **EXPAND** command computes and displays all subitems of a specified item up to a given depth. An item is specified by specifying the complete path starting at one of the root items like “Symbol Table” or “Object Pool”. Names with spaces must be surrounded by double quotes.

To expand all subitems of TargetObject in the Object Pool up to 4 levels, the following command can be used:

```
inspect < ATTRIBUTES EXPAND "Object Pool" TargetObject 4
```

NOTE Because the name Object Pool contains a space, it must be surrounded by double quotes.

NOTE The symbol Table, Stack or other Items may have recursive information. So it may occur that the information tree grows with the depth. Therefore, specifying large expand values may use a large amount of memory.

The **COLLAPSE** command folds one item. The item name must be given. The following command folds the TargetObject:

```
inspect < ATTRIBUTES COLLAPSE "Object Pool" TargetObject
```

The **SELECT** command shows the information of the specified item on the right pane. The following command shows all Objects attached to the TargetObject:

```
inspect < ATTRIBUTES SELECT "Object Pool" TargetObject
```

The **SPLIT** command sets the position of the split line between the left and right pane. The value must be between 0 and 100. A value of 0 only shows the right pane, a value of 100 shows the left pane. Any value between 0 and 100 makes a relative split. The following command makes both panes the same size:

```
inspect < ATTRIBUTES SPLIT 50
```

The **MAXELEM** command sets the number of subitems to display. After the following command, the Inspector will prompt for 1000 subitems:

```
inspect < ATTRIBUTES MAXELEM ON 1000
```

The **FORMAT** command specifies whether integral values like addresses should be displayed as hexadecimal or decimal. The following command specifies the hexadecimal display:

```
inspect < ATTRIBUTES FORMAT Hex
```

Equivalent Operations

ATTRIBUTES COLUMNWIDTH ~ Modify column width with the mouse.

ATTRIBUTES EXPAND ~ Expand any item with the mouse.

ATTRIBUTES COLLAPSE ~ Collapse the specified item with the mouse.

ATTRIBUTES SELECT ~ Click on the specified item to select it.

ATTRIBUTES SPLIT ~ Move the split line between the panes with the mouse.

ATTRIBUTES MAXELEM ~ Select **max. Elements...** from the context menu.

AT

The **AT** command temporarily suspends a command file from executing until after a specified delay in milliseconds. The delay is measured from the time the command file is started. In the event that command files are chained (one calling another), the delay is measured from the time the first command file is started.

NOTE This command can only be executed from a command file. The time specified is relative to the start of command file execution.

Usage

AT **time**

where **time=expression** and **expression** is interpreted in milliseconds**.**

Components

Debugger engine.

Example:

```
AT 10 OPEN Command
```

This command (in command file) opens the **Command Line component** 10 ms after the command file is executed.

AUTOSIZE

AUTOSIZE enables/disables windows autosizing. When on, the size of component windows are automatically adapted to the Simulator/Debugger main window when it is resized.

Usage

AUTOSIZE on|off

Components

Debugger engine.

Example:

```
in>AUTOSIZE off
```

Windows autosizing is disabled.

BASE

In the Profiler component, the **BASE** command sets the profiler base to **code** (total code) or **module** (each module code).

Usage

BASE code|module

Components

Profiler component.

Example:

```
in>BASE code
```

BC

BC deletes a breakpoint at the specified address. When ***** is specified, all breakpoints are deleted.

You can point to the breakpoint in the Assembly or Source component window, right-click and choose **Delete Breakpoint** in the popup menu, or open the ControlPoints Window, select the breakpoint from the list and click **Delete**.

NOTE Correct module names are displayed in the Module component window. Make sure that the module name of your command is correct: if the `.abs` is in **HIWARE** format, some debug information is in the object file (`.o`), and module names have a `.o` extension (e.g., `fibo.o`). In **ELF** format, module name extensions are `.c`, `.cpp` or `.dbg` (`.dbg` for program sources in assembler) (e.g., `fibo.c`), since all debugging information is contained in the `.abs` file and object files are not used. Adapt the following examples with your `.abs` application file format.

Usage

BC address|*

address is the address of the breakpoint to be deleted. This address is specified in ANSI C or standard Assembler format. **address** can also be replaced by an **expression** as shown in the example below.

When ***** is specified all breakpoints are deleted.

Components

Debugger engine.

Example1:

```
in>BC 0x8000
```

This command deletes the breakpoint set at the address 0x8000. The breakpoint symbol is removed in the source and assembly window. The breakpoint is removed from the breakpoint list.

Example2:

```
in>BC &FIBO.C:Fibonacci
```

In this example, an **expression** replaces the address. FIBO.C is the module name and Fibonacci is the function where the breakpoint is cleared.

BCKCOLOR

BCKCOLOR sets the background color.

The background color defined with the BCKCOLOR command is valid for all component windows. Avoid using the same color for the font and background, otherwise text in the component windows will not be visible. Also avoid using colors that have a specific meaning in the command line window. These colors are:

Red: used to display error messages.

Blue: used to echo commands.

Green: used to display asynchronous events.

NOTE When WHITE is given as a parameter, the default background color for all component windows is set, for example, the register component is lightgrey.

Usage

BCKCOLOR color

Where **color** can be one of the following: BLACK, GREY, LIGHTGREY, WHITE, RED, YELLOW, BLUE, CYAN, GREEN, PURPLE, LIGHTRED, LIGHTYELLOW, LIGHTBLUE, LIGHTCYAN, LIGHTGREEN, LIGHTPURPLE

Components

Debugger engine.

Example:

```
in>BCKCOLOR LIGHTCYAN
```

The background color of all currently open component windows is set to Lightcyan. To return to the original display, enter **BCKCOLOR WHITE.**

BD

In the Command Line component, the **BD** command displays the list of all breakpoints currently set with addresses and types (temporary, permanent).

Usage

BD

Components

Debugger engine.

Example:

```
in>BD
Fibonacci 0x805c T
Fibonacci 0x8072 P
Fibonacci 0x8074 T
main 0x8099 T
```

One permanent and two temporary breakpoints are set in the function **Fibonacci**, and one temporary breakpoint is set in the **main** function.

NOTE From the list, it is not possible to know if a breakpoint is disabled or not.

BS

BS sets a temporary (**T**) or a permanent (**P**) breakpoint at the specified address. If no **P** or **T** is specified, the default is a permanent (**P**) breakpoint.

Equivalent Operation

You can point at a statement in the Assembly or Source component window, right-click and choose **Set Breakpoint** in the popup menu, or open the Controlpoints Configuration Window and choose **Show Breakpoint**, then select the breakpoint and set its properties.

NOTE Correct module names are displayed in the Module component window. Make sure that the module name of your command is correct:
If the `.abs` is in **HIWARE** format, some debug information is in the object file (`.o`), and module names have a `.o` extension (e.g., `fibo.o`). In **ELF** format, module name extensions are `.c`, `.cpp` or **.**`dbg` (**.**`dbg` for program sources in assembler) (e.g., `fibo.c`), since all debugging information is contained in the `.abs` file and object files are not used. Adapt the following examples with `.abs` application file format.

Usage

BS address| function [{mark}]

[P|T[state]][;cond=”condition”[state]]

[;cmd=”command”[state]][;cur=current[inter=interval]]

[;cdSz=codeSize[srSz=sourceSize]]

address is the address where the breakpoint is to be set. This address is specified in ANSI C format. **address** can also be replaced by an **expression** as shown in the example below.

function is the name of the function in which to set the breakpoint.

mark (displayed mark in Source component window) is the mark number where the breakpoint is to be set. When mark is:

- > 0: the position is relative to the beginning of the function.
- = 0: the position is the entry point of the function (default value).
- < 0: the position is relative to the end of the function.

P, specifies the breakpoint as a permanent breakpoint.

T, specifies the breakpoint as a temporary breakpoint. A temporary breakpoint is deleted once it is reached.

State is **E** or **D** where **E** is for enabled (state is set by default to **E** if nothing is specified), and **D** is for disabled.

condition is an **expression.** It matches the **Condition** field in the Controlpoints Configuration window for a conditional breakpoint.

command is any Debugger command (at this level, the commands **G**, **GO** and **STOP** are not allowed). It matches the **Command** field in the Controlpoints Configuration window, for associated commands. For the **Command** function, the states are **E (enabled)** or **C (continue)**.

current is an **expression.** It matches the **Current** field (**Counter**) in the Controlpoints Configuration window, for counting breakpoints.

interval is an **expression**. It matches the **Interval** field (**Counter**) in the Controlpoints Configuration window, for counting breakpoints.

codeSize is an **expression**. It is usually a constant number to specify (for security) the code size of a function where a breakpoint is set. If the size specified does not match the size of the function currently loaded in the .ABS file, the breakpoint is set but disabled.

sourceSize is an **expression.** It is usually a constant number to specify (for security) the source (text) size of a function where a breakpoint is set. If the size specified does not match the size of the function in the source file, the breakpoint is set but disabled.

Components

Debugger engine.

Example:

```
in>BS 0x8000 T
```

This command sets a temporary breakpoint at the address 0x8000.

```
in>BS $8000
```

This command sets a permanent breakpoint at the address 0x8000.

```
BS &FIBO.C:Fibonacci
```

In this example, an **expression** replaces the address. `FIBO.C` is the module name and Fibonacci is the function where the breakpoint is set.

More Examples:

```
in>BS &main + 22 P E ; cdSz = 66 srSz = 134
```

Sets a breakpoint at the address of the main procedure + 22, where the code size of the main procedure is 66 bytes and its source size is 134 characters.

```
in>BS Fibo.c:main{3}
```

Sets a breakpoint at the 3rd mark of the procedure **main,** where **main** is a function of the `FIBO.C` module.

```
in>BS &counter + 5; cond ="fib1>fib2";cmd="bckcolor red"
```

Sets a breakpoint at the address of the variable **counter** + 5, where the condition is **fib1 > fib2** and the command is **"bckcolor red"**.

```
in>BS &Fibo.c:Fibonacci+13
```

Sets a breakpoint at the address of the **Fibonacci** procedure + 13, where **Fibonacci** is a function of the `FIBO.C` module.

CALL

Executes a command in the specified command file.

NOTE If no path is specified, the destination directory is the current project directory.

Usage

CALL FileName [;C][;NL]

Components

Debugger engine.

Example:

```
in>cf \util\config.cmd
```

Loads the config command file.

CD

The **CD** command changes the current working directory to the directory specified in path. When the command is entered with no parameter, the current directory is displayed.

The directory specified in the CD command must be a valid directory. It should exist and be accessible from the PC. When specifying a relative path in the CD command, make sure the path is relative to the project directory.

NOTE When no path is specified, the default directory is the project directory. When using the CD command, all commands referring to a file with no path specified could be affected.

Usage

CD [path]

path: The pathname of a directory that becomes the current working directory (case insensitive).

Components

Debugger engine.

Example:

```
in>cd..
C:\Freescale\demo
in>cd
C:\Freescale\demo
in>cd /Freescale/prog
C:\Freescale\prog
```

The new project directory is `C:\Freescale\prog`

CF

The **CF** command reads the commands in the specified command file, which are then executed by the command interpreter. The command file contains ASCII text commands. Command files can be nested. By default, after executing the commands from a nested command file, the command interpreter resumes execution of remaining commands in the calling file. Any error halts execution of **CF** file commands. When the command is entered with no parameter, the **Open File** dialog is displayed. The **CALL** command is equivalent to the **CF** command.

NOTE If no path is specified, the destination directory is the current project directory.

Usage

CF **fileName [;C][;NL]**

Where **fileName** is a file (and path) containing Simulator/Debugger commands.

;C specifies chaining the command file. This option is meaningful in a nested command file only.

When the **;C** option is given in the calling file, the command interpreter quits the calling file and executes the called file. (i.e. in the calling file, commands following the **CF ... ;C** command are never executed).

When the option is omitted, execution of the remaining commands in the calling file is resumed after the commands in the called file have been executed.

;NL: when set, the commands that are in the called file are not logged in the Command Line window (and not to log file, when a file has been opened with an LF on page 696 command), even if the **CMDFILE** type is set to **ON** (see also the LOG on page 699 command).

Components

Debugger engine.

Examples:

```
in>CF commands.txt
```

The `COMMANDS.TXT` file is executed. It should contain debugger commands like those described in this chapter.

Example Without ";C" Option:

if a `command1.txt` file contains:

```
bckcolor green
cf command2.txt
```

```
bckcolor white
```

if a `command2.txt` file contains:

```
bckcolor red
```

Execution:

```
in>cf command1.txt
executing command1.txt

!bckcolor green
!cf command2.txt
executing command2.txt

1!bckcolor red
1!
1!
done command2.txt

!bckcolor white
!
done command1.txt
```

Example With ";C" Option:

if a **command1.txt** file contains:

```
bckcolor green
cf command2.txt ;C
bckcolor white
```

if a **command2.txt** file contains:

```
bckcolor red
```

Execution:

```
in>cf command1.txt
executing command1.txt

!bckcolor green
!cf command2.txt ;C
executing command2.txt

1!bckcolor red
1!
```

```
1!
done command2.txt

done command1.txt
```

CLOCK

In the SoftTrace component, the **CLOCK** command sets the clock speed.

Usage

CLOCK frequency

Where number is a decimal number, which is the CPU frequency in Hertz.

Components

SoftTrace component.

Example:

```
in>CLOCK 4000000
```

CLOSE

The **CLOSE** command is used to close a component.

Component names are: Assembly, Command, Coverage, Data, Inspect, IO_Led, Led, Memory, Module, Phone, Procedure, Profiler, Recorder, Register, SoftTrace, Source, Stimulation.

Usage

CLOSE component | *

where ***** means “all components”.

Components

Debugger engine.

Example:

```
in>CLOSE Memory
```

The Memory component window is closed (unloaded).

COPYMEM

The **COPYMEM** command is used to copy a memory range to a destination range defined by the beginning address. This command works on defined memory only. The source range and destination range are tested to ensure they are not overlayed.

Usage

COPYMEM <Source address range> dest-address

Components

Memory.

Example:

```
in>copymem 0x3FC2A0..0x3FC2B0 0x3FC300
```

The memory from 0x3FC2A0 to 0X3FC2B0 is copied to the memory at 0x3FC300 to 0x3FC310. This Memory range appears in red in the Memory Component.

CMDFILE

The **CMDFILE** command allows you to define all target specific commands in a command file. For example, startup, preload, reset, and path of this file.

Usage

CMDFILE <Command File Kind> ON|OFF ["<Command File Full Name>"]

Components

Simulator/target engine.

Example:

```
in>cmdfile postload on "c:\temp\myposloadfile.cmd"
```

The myposloadfile command file will be executed after loading the absolute file.

CR

The **CR** command initiates writing records of commands to an external file. Writing records continues until a close record file (NOCR on page 708) command is executed.

NOTE Drag & drop actions are also translated into commands in the record file.

NOTE If no path is specified, the destination directory is the current project directory.

Usage

CR [fileName][;A]

If fileName is not specified, a standard **Open File** dialog is opened.

;A specifies to open a file **fileName** in append mode. Records are appended at the end of an existing record file.

If the **;A** option is omitted and **fileName** is an existing file, the file is cleared before records are written to it.

Components

Debugger engine.

Example:

```
in>cr /Freescale/demo/myrecord.txt ;A
```

The `myrecord.txt` file is opened in “Append” mode for a recording session.

CYCLE

In the **SoftTrace component**, the **CYCLE** command displays or hides cycles. When cycle is off, milliseconds (ms) are displayed.

Usage

CYCLE on|off

Components

Softtrace component.

Example:

```
in>CYCLE on
```

DASM

The **DASM** command displays the assembler code lines of an application, starting at the address given in the parameter. If there is no parameter, the assembler code following the last address of the previous display is displayed.

This command can be stopped by pressing the **Esc** key.

Equivalent Operation

Right-click in the Assembly component window, select **Address...** and enter the address to start disassembly in the **Show PC** dialog.

Usage

DASM [**address**|**range**][**;OBJ**]

address: A constant expression representing the **address** where disassembly begins.

range: An address range constant that specifies addresses to be disassembled. When **range** is omitted, a maximum of sixteen instructions are disassembled.

When **address** and **range** are omitted, disassembly begins at the address of the instruction that follows the last instruction that has been disassembled by the most recent **DASM** command. If this is the first **DASM** command of a session, disassembly begins at the current address in the program counter.

;OBJ: Displays assembler code in hexadecimal.

Components

Debugger engine.

Example:

```
in>dasm 0xf04b
00F04B LDHX   #0x0450
00F04E TXS
00F04F CLRH
00F050 CLRX
00F051 STX    0x80
00F053 INC    0x80
00F055 LDX    0x80
00F057 JSR    0xF000
00F05A STX    0x82
00F05C STA    0x81
00F05E LDA    #0x17
00F060 CMP    0x80
00F062 BEQ    *-20       /abs = F050
00F064 BRA    *-19       /abs = F053
```

```
00F066 DECX
00F067 DECX
```

NOTE Depending on the target, the above code may vary.

Disassembled instructions are displayed in the Command Line component window. Therefore, it is necessary to open the Command Line component before executing this command to see the dumped code.

DB

The **DB** command displays the hexadecimal and ASCII values of the bytes in a specified range of memory. The command displays one or more lines, depending on the address or range specified. Each line shows the address of the first byte displayed in the line, followed by the number of specified hexadecimal byte values. The hexadecimal byte values are followed by the corresponding ASCII characters, separated by spaces. Between the eighth and ninth values, a hyphen (-) replaces the space as the separator. Each non-displayable character is represented by a period (.).

This command can be stopped by pressing the **Esc** key.

Usage

DB [**address**|**range]**

When **address** and **range** are omitted, the first longword displayed is taken from the address following the last longword displayed by the most recent **DB**, **DW**, or **DL** command, or from address **0x0000** (for the first **DB**, DW on page 675, DL on page 674 command of a session).

Components

Debugger engine.

Examples:

```
in>DB 0x8000..0x800F
```

8000: FE 80 45 FD 80 43 27 10-35 ED 31 EC 31 69 70 83 þ_Eý_C'.5í1ì1ipƒ

Memory bytes are displayed in the Command Line component window, with matching ASCII characters. So, it is necessary to open the Command Line component before executing this command to see the dumped code.

```
in>DB &TCR
```

0012: 5A Z

displays the byte that is at the address of the TCR I/O register. I/O registers are defined in a `DEFAULT.REG` file.

DDEPROTOCOL

The **DDEPROTOCOL** command is used to configure the Debugger/Simulator dynamic data exchange (DDE) protocol.

By default the DDE protocol is activated and not displayed in the command line component.

Usage

DDEPROTOCOL ON|OFF|SHOW|HIDE|STATUS

Where:

- ON enables the DDE communication protocol
- OFF disables the DDE communication protocol
- SHOW displays DDE protocol information in the command line component
- HIDE hides DDE protocol information in the command line component
- STATUS provides information if the DDE protocol is active (on or off) and if display is active (Show or Hide)

Components

Debugger engine.

Example:

```
in>DDEPROTOCOL ON
in>DDEPROTOCOL SHOW
in>DDEPROTOCOL STATUS
DDEPROTOCOL ON - DISPLAYING ON
```

The DDE protocol is activated and displayed, and status is given in the command line component.

NOTE For more information on Debugger/Simulator DDE implementation, please refer to the chapter that deals with Debugger DDE Capabilities on page 209.

DEFINE

The **DEFINE** command creates a symbol and associates the value of an expression with it. Arithmetic expressions are evaluated when the command is interpreted. The symbol can be used to represent the expression until the symbol is redefined, or undefined using the **UNDEF** command. A symbol is a maximum of 31 characters long. In a command line, all symbol occurrences (after the command name) are substituted by their values before processing starts. A symbol cannot represent a command name. Note that a symbol definition precedes (and hence conceals) a program variable with the same name.

Defined symbols remain valid when a new application is loaded. An application variable or I/O register can be overwritten with a **DEFINE** command.

NOTE This command can be used to assign meaningful names to expressions, which can be used in other commands. This increases the readability of command files and avoids re-evaluation of complex expressions.

Usage

DEFINE symbol [=] expression

Components

Debugger engine.

Example:

```
in>DEFINE addr $1000
in>DEFINE limit = addr + 15
```

First addr is defined as a constant equivalent to $1000. Then limit is defined and affected with the value ($1000 + 15)

A symbol defined in the loaded application can be redefined on the command line using the **DEFINE** command. The symbol defined in the application is not accessible until an **UNDEF** on that symbol name is detected in the command file.

Example:

A symbol named ‘testCase’ is defined in the test application.

```
/* Loads application test.abs */
LOAD   test.abs
/* Display value of testCase. */
DB     testCase
/* Redefine symbol testCase. */
DEFINE testCase = $800
/*Display value stored at address $800.*/
DB     testCase
/* Redefine symbol testCase. */
UNDEF testCase
/* Display value of testCase. */
DB     testCase
```

NOTE Also refer to examples given for the command UNDEF on page 735.

DETAILS

In the **Profiler component**, the **DETAILS** command opens a profiler split view in the Source or Assembly component.

Usage

DETAILS assembly|source

Components

Profiler components.

Example:

```
in>DETAILS source
```

DL

The **DL** command displays the hexadecimal values of the longwords in a specified range of memory. The command displays one or more lines, depending on the address or range specified. Each line shows the address of the first longword displayed in the line, followed by the number of specified hexadecimal longword values.

When a size is specified in the range, this size represents the number of longwords that should be displayed in the command line window.

This command can be stopped by pressing the **Esc** key.

NOTE Open the Command Line component before executing this command to see the dumped code.

Usage

DL [address|**range]**

When **range** is omitted, the first longword displayed is taken from the address following the last longword displayed by the most recent **DB**, **DW**, or **DL** command, or from address **0x0000** (for the first DB on page 669, DW on page 675, DL on page 674 command of a session).

Components

Debugger engine.

Example:

```
in>DL 0x8000..0x8007
```

8000: FE8045FD 80432710

The content of the memory range starting at 0x8000 and ending at 0x8007 is displayed as longword (4-bytes) values.

```
in>DL 0x8000,2
```

8000: FE8045FD 80432710

The content of 2 longwords starting at 0x8000 is displayed as longword (4-bytes) values.

Memory longwords are displayed in the Command Line component window.

DUMP

The DUMP command writes everything visible in the Data component to the command line component.

Usage

DUMP

Components

Data component.

Example:

```
in> Data:1 < DUMP
```

DW

The **DW** command displays the hexadecimal values of the words in a specified range of memory. The command displays one or more lines, depending on the address or range specified. Each line shows the address of the first word displayed in the line, followed by the number of specified hexadecimal word values.

When a size is specified in the range, this size represents the number of words that should be displayed in the command line window.

This command can be stopped by pressing the **Esc** key.

NOTE Open the Command Line component before executing this command to see the dumped code.

Usage

DW [address | **range]**

When **address** is an address constant expression, the address of the first word is displayed.

When **address** and **range** are omitted, the first word displayed is taken from the address following the last word displayed by the most recent **DB**, **DW**, or **DL** command, or from address **0x0000** (for the first DB on page 669, DW on page 675, DL on page 674 command of a session).

Components

Debugger engine.

Example:

```
in>DW 0x8000,4
```

8000: FE80 45FD 8043 2710

The content of 4 words starting at 0x8000 is displayed as word (2-bytes) values.

Memory words are displayed in the Command Line component window.

E

The **E** command evaluates an expression and displays the result in the Command Line component window. When the expression is the only parameter entered (no option specified) the value of the expression is displayed in the default number base. The result is displayed as a signed number in decimal format and as unsigned number in all other formats.

Usage

E expression[;O|D|X|C|B]

where:

;O: displays the value of expression as an octal (base 8) number.

;D: displays the value of expression as a decimal (base 10) number.

;X: displays the value of expression as an hexadecimal (base 16) number.

;C: displays the value of expression as an ASCII character. The remainder resulting from dividing the number by 256 is displayed. All values are displayed in the current font. Control characters (<32) are displayed as decimal.

;B: displays the value of expression as a binary (base 2) number.

Components

Debugger engine.

Example:

```
in>define a=0x12
in>define b=0x10
in>e a+b
in>=34
```

The addition operation of the two previously defined variables **a** and **b** is evaluated and the result is displayed in the Command Line window. The output can be redirected to a file by using the **LF** command (refer to LF on page 696 and LOG on page 699 command descriptions).

ELSE

The **ELSE** keyword is associated with the LF on page 696 command.

Usage

ELSE

Components

Debugger engine.

Example:

```
if CUR_TARGET == 1000          /* Condition */
      set sim
else set bdi                  /* Other Condition */
```

ELSEIF

The **ELSEIF** keyword is associated with the IF on page 694 command.

Usage

ELSEIF condition

where **condition** is same as defined in “C” language.

Components

Debugger engine.

Example:

```
if CUR_TARGET == 1000          /* Simulator */
      set sim
elseif CUR_TARGET == 1001  /* BDI */
      set bdi
```

ENDFOCUS

The **ENDFOCUS** command resets the current focus. It is associated with the **FOCUS** command. Following commands are broadcast to all currently open components. This command is only valid in a command file.

Usage

ENDFOCUS

Components

Debugger engine.

Example:

```
FOCUS Assembly
ATTRIBUTES code on
ENDFOCUS
FOCUS Source
ATTRIBUTES marks on
ENDFOCUS
```

The ATTRIBUTES command is first redirected to the Assembly component by the **FOCUS** Assembly command. The code is displayed next to assembly instructions. Then the Assembly component is released by the ENDFOCUS command and the second ATTRIBUTES command is redirected to the Source component by the **FOCUS** Source command. Marks are displayed in the Source window.

ENDFOR

Description

The **ENDFOR** keyword is associated with the FOR on page 687 command.

Usage

ENDFOR

Components

Debugger engine.

Example:

```
for i = 1..5
   define multi5 = 5 * i
endfor
```

After the **ENDFOR** instruction, i is equal to 5.

ENDIF

Description

The **ENDIF** keyword is associated with the IF on page 694 command.

Usage

ENDIF

Components

Debugger engine.

Example:

```
if (CUR_CPU == 12)
  DW &counter
else
  DB &counter
endif
```

ENDWHILE

Description

The **ENDWHILE** keyword is associated with the WHILE on page 742 command.

Usage

ENDWHILE

Components

Debugger engine.

Example:

```
while i < 5
  define multi5 = 5 * i
  define i = i + 1
endwhile
```

After the **ENDWHILE** instruction, i is equal to 5

EXECUTE

Description

In the Stimulation component, the **EXECUTE** command executes a file containing stimulation commands. Refer to the **I/O Stimulation** document.

Usage

EXECUTE fileName

Components

Stimulation component.

Example:

```
in>EXECUTE stimu.txt
```

EXIT

Description

In the Command line component, the **EXIT** command closes the Debugger application.

Usage

EXIT

Components

Debugger engine.

Example:

in>EXIT

The Debugger application is closed.

FILL

Description

In the Memory component, the **FILL** command fills a corresponding range of Memory component with the defined value. The value must be a single byte pattern (higher bytes ignored).

Usage

FILL range value

the syntax for range is: LowAddress..HighAddress

Components

Memory component.

Equivalent Operation

The **File Memory** dialog is available from the Memory popup menu and by selecting **Fill...** or **Memory>Fill...** menu entry.

Example:

```
in>FILL 0x8000..0x8008 0xFF
```

The memory range 0x8000..0x8008 is filled with the value 0xFF.

FILTER

Description

In the Memory component, with the FILTER command, you select what you want to display, for example **modules**: modules only, **functions**: modules and functions, or **lines**: modules and functions and code lines. You can also specify a range to be logged in your file. **Range** must be between 0 and 100.

Usage

FILTER Options [<range>]

Options = modules|functions|lines

Components

Coverage component.

Example:

```
in>coverage < FILTER functions 25..75
```

FIND

Description

In the Source component, the **FIND** command is used to search a specified pattern in the source file currently loaded. If the pattern has been found, it is highlighted. The search is forward (default), backward (**;B**), match case sensitive (**;MC**) or match whole word sensitive (**;WW**). The operation starts form the currently highlighted statement or from the beginning of the file (if nothing is highlighted). If the item is found, the Source window is scrolled to the position of the item and the item is highlighted in grey.

Equivalent Operation

You can select **Source>Find...** or open the Source popup menu and select **Find...** to open the **Find** dialog.

Usage

FIND **"**string" [;B] [;MC] [;WW]

Where **string** is the "**pattern**" to match. It has to be enclosed in double quotes. See the example below.

;B the search is backwards, default is forwards.

;MC match case sensitive is set.

;WW match whole word is set.

Components

Source component.

Example:

```
in>FIND "this" ;B ;WW
```

The "**this**" string (considered as a whole word) is searched in the Source component window. The search is performed backward.

FINDPROC

Description

If a valid procedure name is given as parameter, the source file where the procedure is defined is opened in the Source Component. The procedure's definition is displayed and the procedure's title is highlighted.

Equivalent Operation

You can select **Source>Find Procedure...** or open the Source popup menu and select **Find Procedure...** to open the **Find Procedure** dialog.

Usage

FINDPROC procedureName

Components

Source component.

Example:

```
in>findproc Fibonacci
```

The "**Fibonacci**" procedure is displayed and the title is highlighted.

FOCUS

Description

The **FOCUS** command sets the given component (**component**) as the destination for all subsequent commands up to the next ENDFOCUS on page 678 command. Hence, the focus command releases the user from repeatedly specifying the same command redirection, especially in the case where command files are edited manually. This command is only valid in a command file.

NOTE It is not possible to visually notice that a component is “FOCUSed”. However, you can use the ACTIVATE on page 644 command to activate a component window.

Usage

FOCUS component

Components

Debugger engine.

Example:

```
FOCUS Assembly
ATTRIBUTES code on
ENDFOCUS
FOCUS Source
ATTRIBUTES marks on
ENDFOCUS
```

The ATTRIBUTES command is first redirected to the Assembly component by the **FOCUS** Assembly command. The code is displayed next to assembly instructions. Then the Assembly component is released by the ENDFOCUS command and the second ATTRIBUTES command is redirected to the Source component by the **FOCUS** Source command. Marks are displayed in the Source window.

FOLD

Description

In the Source component, the **FOLD** command hides the source text at the program block level. Folded program text is displayed as if the program block was empty. When the folded block is unfolded, the hidden program text reappears. All text is folded once or (*****) completely, until there are no more folded parts.

Usage

FOLD [*]

Where ***** means fold completely, otherwise fold only one level.

Components

Source component.

Example:

```
in>FOLD *
```

FONT

Description

FONT sets the font type, size and color.

Equivalent Operation

The **Font** dialog is available by selecting the **Component>Fonts...** menu entry.

Usage

FONT ‘FontName’ [size][color]

Components

Debugger engine.

Example:

```
FONT ‘Arial’ 8 BLUE
```

The font type is “Arial” 8 points and blue.

FOR

Description

The **FOR** loop allows you to execute all commands up to the trailing ENDFOR on page 679 a predefined number of times. The bounds of the range and the optional steps are evaluated at the beginning. A **variable** (either a symbol or a program variable) may be optionally specified, which is assigned to all values of the range that are met during execution of the for loop. If a variable is used, it must be defined before executing the **FOR** command, with a DEFINE on page 672 command.

Assignment happens immediately before comparing the iteration value with the upper bound. The variable is only a copy of the internal iteration value, therefore modifications on the variable don't have an impact on the number of iterations.

This command can be stopped by pressing the **Esc** key.

Usage

FOR[variable =]range [“,” step]

Where **variable** is the name of a defined variable.

range: This is an address range constant that specifies addresses to be disassembled.

step: constant number matching the step increment of the loop.

Components

Debugger engine.

Example:

```
DEFINE loop = 0
FOR loop = 1..6,1
T
ENDFOR
```

The T Trace command is performed 6 times.

FPRINTF

Description

FPRINTF is the standard ANSI C command: Writes formatted output string to a file.

Usage

FPRINTF (<filename>, <&format>, <expression>, <expression>, ...)

Components

Debugger engine.

Example:

```
fprintf (test.txt,"%s %2d","The value of the counter
is:",counter)
```

The content of the file `test.txt` is: The value of the counter is: 25

FRAMES

Description

In the **SoftTrace component**, the **FRAMES** command sets the maximum number of frame records.

Usage

FRAMES number

Where **number** is a decimal number, which is the maximum number of recorded frames. This number must not exceed 1000000.

Components

SoftTrace component.

Example:

```
FRAMES 10000
```

G

Description

The **G** command starts code execution in the emulated system at the current address in the program counter or at the specified address. You can therefore specify the entry point of your program, skipping execution of the previous code.

Usage

G [address]

When no **address** is entered, the address in the program counter is not altered and execution begins at the address in the program counter.

Alias

GO

Components

Debugger engine.

Example:

```
G 0x8000
```

Program execution is started at 0x8000. **RUNNING** is displayed in the status bar. The application runs until a breakpoint is reached or you stop the execution.

GO

Description

The **GO** command starts code execution in the emulated system at the current address in the program counter or at the specified address. You can therefore specify the entry point of your program, skipping execution of previous code.

Usage

GO [address]

When no **address** is entered, the address in the program counter is not altered and execution begins at the address in the program counter.

Alias

G

Components

Debugger engine.

Example:

```
in>GO 0x8000
```

Program execution is started at address 0x8000. **RUNNING** is displayed in the status bar. The application runs until a breakpoint is reached or you stop execution.

GOTO

Description

The **GOTO** command diverts execution of the command file to the command line that follows the Label. The Label must be defined in the current command file. The **GOTO** command fails, if the Label is not found. A label can only be followed on the same line by a comment.

Usage

GOTO **Label**

Components

Debugger engine.

Example:

```
GOTO MyLabel
...
...
MyLabel:  // comments
```

When the instruction GOTO MyLabel is reached, the program pointer jumps to MyLabel and follows program execution from this position.

GOTOIF

Description

The **GOTOIF** command diverts execution of the command file to the command line that follows the label if the condition is true. Otherwise, the command is ignored. The **GOTOIF** command fails, if the condition is true and the label is not found.

Usage

GOTOIF condition Label

where condition is same as defined in “C” language.

Components

Debugger engine.

Example:

```
DEFINE jump = 0
...
DEFINE jump = jump + 1
...
GOTOIF  jump == 10 MyLabel
T
...
MyLabel:  // comments
```

The program pointer jumps to MyLabel only if jump equals 10. Otherwise, the next instruction (T Trace command) is executed.

GRAPHICS

Description

In the Profiler component, **GRAPHICS** switches the percentages display in the graph bar **on/off**.

Usage

GRAPHICS on|off

Components

Profiler component.

Example:

```
in>GRAPHICS off
```

HELP

In the Command line component, the **HELP** command displays all available commands.

Subcommands from the **ATTRIBUTES** command are not listed.

Component specific commands, which are not open, will not be listed either.

Usage

HELP

Components

Debugger engine.

Example:

```
in>HELP
```

```
HI-WAVE Engine:
  VER
  LF
  NOLF
  CR
  NOCR
   ....
```

IF

The conditional commands (IF on page 694, ELSEIF on page 677, ELSE on page 677 and ENDIF on page 679) allow you to execute different sections depending on the result of the corresponding condition. The conditional command may be nested. Conditions of the **IF** and **ELSEIF** commands, respectively, guard all commands up to the next **ELSEIF**, **ELSE** or **ENDIF** command on the same nesting level. The **ELSE** command guards all commands up to the next **ENDIF** command on the same nesting level. Any occurrence of a subcommand not in sequence of **"IF**, zero or more **ELSEIF**, zero or one **ELSE**, **ENDIF"** is an error.

Usage

IF condition

Where **condition** is same as defined in "C" language.

Components

Debugger engine.

Example:

```
DEFINE jump = 0
...
DEFINE jump = jump + 1
...
IF  jump == 10
  T
  DEFINE jump = 0
ELSEIF jump == 100
  DEFINE jump = 1
ELSE
  DEFINE jump = 2
ENDIF
```

The jump = = 10 condition is evaluated and depending on the test result, the T Trace instruction is executed, or the ELSEIF jump = = 100 test is evaluated.

INSPECTOROUTPUT

The Inspector dumps the content of the specified item and all computed subitems to the command window. Uncomputed subitems are not printed. To compute all information, the **ATTRIBUTES EXPAND** command is used.

Usage

INSPECTOROUTPUT [name {subname}]

The **name** specifies any of the root items. The **subname** specifies a recursive path to subitems.

If a name contains a space, it must be surrounded by double quotes (").

Components

Inspector component.

Example:

```
in>loadio swap
in>Inspect<ATTRIBUTES EXPAND 3
in>INSPECTOROUTPUT "Object Pool" Swap
```

```
Swap
*  Name      Value  Address  Init...
-  IO_Reg_1  0x0    0x1000   0x0 ...
-  IO_Reg_2  0x0    0x1001   0x0 ...
```

INSPECTORUPDATE

The Inspector displays various information. Some types of information are automatically updated. To make sure that displayed values correspond to the current situation, the **INSPECTORUPDATE** command updates all information.

Usage

INSPECTORUPDATE

Components

Inspector component.

Example:

```
in>INSPECTORUPDATE
```

LF

The **LF** command initiates logging of commands and responses to an external file or device. While logging remains in effect, any line that is appended to the command window is also written to the log file.

Logging continues until a close log file (NOLF on page 708) command is executed. When the **LF** command is entered with no filename, the Open File Dialog is displayed to specify a filename.

Use the logging option (LOG on page 699) command to specify information to be logged.

If a path is specified in the file name, this path must be a valid path. When a relative path is specified, ensure that the path is relative to the project directory.

Usage

LF [fileName][;A]

fileName is a DOS filename that identifies the file or device where the log is written. The command interpreter does not assume a filename extension.

;A opens the file in append mode. Logged lines are appended at the end of an existing log file.

If the **;A** option is omitted and **fileName** is an existing file, the file is cleared before logging begins.

Components

Debugger engine.

Example

```
in>lf /mcuez/demo/logfile.txt ;A
```

The logfile.txt file is opened as a Log File in “append” mode.

NOTE If no path is specified, the destination directory is the current project directory.

LOAD

The **LOAD** command loads a framework application (`.abs` file) for a debugging session. When no application name is specified, the **LoadObjectFile** dialog is opened.

If no target is installed, the following error message is displayed:

"Error: no target is installed"

If no target is connected, the following error message is displayed:

"Error: no target is connected"

Usage

LOAD[applicationName] [CODEONLY|SYMBOLSONLY] [NOPROGRESSBAR] [NOBPT] [NOXPR] [NOPRELOADCMD] [NOPOSTLOADCMD] [DELAY] [VERIFYFIRST|VERIFYALL|VERIFYONLY] [VERIFYOPTIONS|SYMBOLSOPTIONS]

Where:

- **applicationName** is the name of the application to load
- **CODEONLY** and **SYMBOLSONLY** loads only the code or symbols
- **NOPROGRESSBAR** loads the application without progress bar
- **NOBPT** loads the application without loading breakpoints file (with BPT extension)
- **NOXPR** loads the application without playing Expression file (with XPR extension)
- **NOPRELOADCMD** loads the application without playing PRELOAD file
- **NOPOSTLOADCMD** loads the application without playing POSTLOAD file
- **DELAY** loads the application and waits one second
- **VERIFYFIRST** matches the "First bytes only" code verification option.
- **VERIFYALL** matches the "All bytes" code verification option.
- **VERIFYONLY** matches the "Read back only" code verification option.
- **VERIFYOPTIONS** displays the "Code Verification Options" group in the "Load Executable File" dialog. If this option is missing, the group is not displayed. However, the verification mode can still be specified with options above.
- **SYMBOLSOPTIONS** displays the "Load Options" group in the "Load Executable File" dialog. If this option is missing, the group is not displayed. However, the code+symbols mode can still be specified with options CODEONLY and SYMBOLSONLY.

NOTE By default, the LOAD command is "code+symbols" with no verification.

NOTE If the "SYMBOLSONLY" parameter is passed, verification parameters are ignored and NO verification is performed.

Components

Debugger engine.

Example:

```
LOAD FIBO.ABS
```

The `FIBO.ABS` application is loaded.

NOTE If no path is specified, the destination directory is the current project directory.

LOADCODE

This command loads code into the target system. This command can be used if no debugging is needed. If no target is installed, the following error message is displayed:

"Error: no target is installed"

If no target is connected, the following error message is displayed:

"Error: no target is connected"

Usage

LOADCODE [applicationName]

Components

Debugger engine.

Example:

```
LOADCODE FIBO.ABS
```

Code of the `FIBO.ABS` application is loaded.

NOTE If no path is specified, the destination directory is the current project directory.

LOADSYMBOLS

This command is similar to the **LOAD** command but only loads debugging information into the debugger. This can be used if the code is already loaded into the target system or programmed into a non-volatile memory device.

If no target is installed, the following error message is displayed:

“Error: no target is installed”

If no target is connected, the following error message is displayed:

“Error: no target is connected”

Usage

LOADSYMBOLS [applicationName]

Components

Debugger engine.

Example:

```
LOADSYMBOLS FIBO.ABS
```

Debugging information of the `FIBO.ABS` application is loaded. If no path is specified, the destination directory is the current project directory.

LOG

The **LOG** command enables or disables logging of information in the Command Line component window (and to logfile, when it as been opened with an LF on page 696 command). If **LOG** is not used, all types are **ON** by default i.e. all information is logged in the Command Line component and log file.

NOTE - **about RESPONSES**: Responses are results of commands. For example, for the DB command, the displayed memory dump is the response of the command. Protocol messages are not responses. - **about ERRORS**: Errors are displayed in red in Command Line component. Protocol messages are not errors. - **about NOTICES**: Notices are displayed in green in the Command Line.

Usage

LOG type [=] state {[,] type [=] state}

Where **type** is one of the following types:

CMDLINE: Commands entered on the command line.

CMDFILE: Commands read from a file.

RESPONSES: Command output response.

ERRORS: Error messages.

NOTICES: Asynchronous event notices, such as breakpoints.

Where **state** is **on** or **off.**

state is the new state of **type**:
When **ON**, enables logging of the type.
When **OFF**, disables logging of the **type**.

Components

Debugger engine.

Example:

```
LOG ERRORS = OFF, CMDLINE = on
```

Error messages are not recorded in the Log File. Commands entered in the Command Line component window are recorded.

More About Logging of IF, FOR, WHILE and REPEAT

When commands executed from a command file are logged, all executed commands that are in a **IF** block are logged. That is, a command file executed with the **CF** or **CALL** command without the **NL** option and with **CMDFILE** flag of the **LOG** command set to **TRUE**. All commands in a block that are not executed because the corresponding condition is false are also logged but preceded with a "-".

Example:

When executing the following command file:

```
define truth = 1
IF truth
  bckcolor blue
  at 2000 bckcolor white
else
  bckcolor yellow
  at 1000 bckcolor white
ENDIF
```

The following log file is generated:

```
!define truth = 1
!IF truth
!  bckcolor blue
!  at 2000 bckcolor white
!else
!-  bckcolor yellow
```

```
!-  at 1000 bckcolor white
!ENDIF
```

When commands executed from a command file are logged, all executed commands that are in the **FOR** loop are logged the number of times they have been executed. That is, a command file executed with the **CF** or **CALL** command without the **NL** option and with the **CMDFILE** flag of the **LOG** command set to **TRUE**.

Example2:

When executing the following file:

```
define i = 1
FOR i = 1..3
   ls
ENDFOR
```

The following log file is generated:

```
!define i = 1
!FOR i = 1..3
!   ls
i             0x1 (1)
!ENDFOR
!   ls
i             0x2 (2)
!ENDFOR
!   ls
i             0x3 (3)
!ENDFOR
```

When commands executed from a command file are logged, all executed commands that are in the **WHILE** loop are logged the number of times they have been executed. That is, a command file executed with the **CF** or **CALL** command without the **NL** option and with the **CMDFILE** flag of the **LOG** command set to **TRUE**.

Example3:

When executing the following file:

```
define i = 1
WHILE i < 3
   define i = i + 1
ls
ENDWHILE
```

The following log file is generated:

```
!define i = 1
!WHILE i < 3
!   define i = i + 1
! ls
i             0x2 (2)
!ENDWHILE
!   define i = i + 1
! ls
i             0x3 (3)
!ENDWHILE
```

When commands executed from a command file are logged, all executed commands that are in the **REPEAT** loop are logged the number of times they have been executed. That is, a command file executed with the **CF** or **CALL** command without the **NL** option and with the **CMDFILE** flag of the **LOG** command set to **TRUE**.

Example4:

When executing the following file:

```
define i = 1
REPEAT
   define i = i + 1
ls
UNTIL i == 4
```

The following log file is generated:

```
repeat
until condition
!define i = 1
!REPEAT
!   define i = i + 1
! ls
i             0x2 (2)
!UNTIL i == 4
!   define i = i + 1
! ls
i             0x3 (3)
!UNTIL i == 4
!   define i = i + 1
! ls
i             0x4 (4)
!UNTIL i == 4
```

LS

In the Command Line window, the **LS** command lists the values of symbols defined in the symbol table and by the user. There is no limit to the number of symbols that can be listed. The size of memory determines the symbol table size. Use the DEFINE on page 672 command to define symbols, and the UNDEF on page 735 command to delete symbols.

The symbols that are listed with the LS command are split in two parts: Applications Symbols and User Symbols.

Usage

LS [symbol | *][;C|S]

Where **symbol** is a restricted regular expression that specifies the symbol whose values are to be listed.

***** specifies to list all symbols.

;C specifies to list symbols in canonical format, which consists of a DEFINE command for each symbol.

;S specifies to list symbol table statistics following the list of symbols.

Components

Debugger engine.

Example:

```
in>ls
```

```
User Symbols:
j             0x2 (2)
Application Symbols:
counter       0x80 (128)
fiboCount     0x81 (129)
j             0x83 (131)
n             0x84 (132)
fib1          0x85 (133)
fib2          0x87 (135)
fibo          0x89 (137)
Fibonacci     0xF000 (61440)
Entry         0xF041 (61505)
```

When LS is performed on a single symbol (e.g., **in>ls counter**) that is an application variable as well as a user symbol, the application variable is displayed.

Example with **j** being an application symbol as well as a user symbol:

```
in>ls j
```

```
Application Symbol:
j             0x83 (131)
```

MEM

The **MEM** command displays a representation of the current system memory map and lower and upper boundaries of the internal module that contains the MCU registers.

Usage

MEM

Components

Debugger engine.

Example:

```
in>mem
```

```
Type    Addresses   Comment
-------------------------------------------------------
IO        0..  3F  PRU or TOP   TOP board resource or the PRU
NONE     40..  4F  NONE
RAM      50.. 64F  RAM
NONE     650.. 7FF  NONE
EEPROM   800.. A7F  EEPROM
NONE     A80..3DFF  NONE
ROM     3E00..FDFF  ROM
IO      FE00..FE1F  PRU or TOP   TOP board resource or the PRU
NONE    FE20..FFDB  NONE
ROM     FFDC..FFFE  ROM
COP     FFFF..FFFF  special ram for cop
RT MEM    0.. 3FF  (enabled)
-------------------------------------------------------
```

MS

The **MS** command sets a specified block of memory to a specified list of byte values. When the **range** is wider than the **list** of byte values, the **list** of byte values is repeated as many times as necessary to fill the memory block.

When the **range** is not an integer multiple of the length of the **list**, the last copy of the **list** is truncated appropriately. This command is identical to the write bytes (WB on page 741) command.

Usage

MS range list

range: is an address range constant that defines the block of memory to be set to the values of the bytes in the list.

list: is a list of byte values to be stored in the block of memory.

Components

Debugger engine.

Example:

```
in>MS 0x1000..0x100F 0xFF
```

The memory range between addresses 0x1000 and 0x100F is filled with the 0xFF value.

NB

Description

The **NB** command changes or displays the default number **base** for the constant values in expressions. The initial default number base is 10 (decimal) and can be changed to 16 (hexadecimal), 8 (octal), 2 (binary) or reset to 10 with this command. The base is always specified as a decimal constant.

Independent of the default base number, the ANSI C standard notation for constant is supported inside an expression. That means that independent of the current number base you can specify hexadecimal or octal constants using the standard ANSI C notation shown in Table 28.6 on page 706.

Usage

NB [base]

Where **base** is the new number base (2, 8, 10 or 16).

Components

Debugger engine.

Table 28.6 ANSI C Constant Notation

Notation	Meaning
0x----	Hexadecimal constant
0----	Octal constant

Table Example:

0x2F00, /* Hexadecimal Constant */

043, /* Octal Constant */

255 /* Decimal Constant */

In the same way, the **Assembler** notation for constant is also supported. That means that independent of the current number base you can specify hexadecimal, octal or binary constants using the **Assembler** prefixes shown in Table 28.7 on page 707.

Table 28.7 Assembler Notation for Constant

Notation	Meaning
$----	Hexadecimal constant
@----	Octal constant
%----	Binary constant

Table Example:

$2F00, /* Hexadecimal Constant */

@43, /* Octal Constant */

%10011 /* Binary Constant */

When the default number base is 16, constants starting with a letter A, B, C, D, E or F must be prefixed either by 0x or by $, as shown in Table 28.8 on page 707. Otherwise, the command line interpreter cannot detect if you are specifying a number or a symbol.

Table 28.8 Base is 16: Constants Starting with Letter A, B, C, D, E or F

Notation	Meaning
5AFD	Hexadecimal constant $5AFD
AFD	Hexadecimal constant $AFD

Table Example:

```
in>NB 16
```

The number base is hexadecimal.

NOCR

The **NOCR** command closes the current record file. The record file is opened with the CR on page 667 command.

Usage

NOCR

Components

Debugger engine.

Example:

```
in>NOCR
```

The current record file is closed.

NOLF

The **NOLF** command closes the current Log File. The log file is opened with the LF on page 696 command.

Usage

NOLF

Components

Debugger engine.

Example:

```
in>NOLF
```

The current Log File is closed.

OPEN

The **OPEN** command is used to open a window component.

Usage

OPEN "component" [x y width height][;I | ;MAX]

where:

- component is the component name with an optional path
- **x** is the X-axis of the upper left corner of the window component
- **y** is the Y-axis of the upper left corner of the window component
- **width** is the width of the window component
- **height** the height of the window component

When **I** is specified, the component window will be iconized; when **MAX** is specified, the component window will be maximized.

Component names are: Assembly, Command, Coverage, Data, Inspect, IO_Led, Led, Memory, Module, Phone, Procedure, Profiler, Recorder, Register, SoftTrace, Source, Stimulation.

Components

Debugger engine.

Example:

```
in>OPEN Terminal 0 78 60 22
```

The Terminal component and window is opened at specified positions and with specified width and height.

OUTPUT

With **OUTPUT**, you can redirect the Coverage component results to an output file indicated by the file name and his path.

Usage

OUTPUT FileName

Where FileName is file name (path + name).

Components

Coverage component.

Example:

```
in>coverage < OUTPUT c:\Freescale\myfile.txt
```

The Coverage output results are redirected to the file `myfile.txt` from the directory `C:\Freescale`.

P

The **P** command executes a CPU instruction, either at a specified address or at the current instruction, (pointed to by the program counter). This command traces through subroutine calls, software interrupts, and operations involving the following instructions (two are target specific):

- Branch to SubRoutine (**BSR**)
- Long Branch to Subroutine (**LBSR**)
- Jump to Subroutine (**JSR**)
- Software Interrupt (**SWI**)
- Repeat Multiply and Accumulate (**RMAC**)

For example: if the current instruction is a **BSR** instruction, the subroutine is executed, and execution stops at the first instruction after the **BSR** instruction. For instructions that are not in the above list, the P on page 711 and T on page 733 commands are equivalent.

When the instruction specified in the **P** command has been executed, the software displays the content of the CPU registers, the instruction bytes at the new value of the program counter and a mnemonic disassembly of that instruction.

Usage

P [address]

address: is an address constant expression, the address at which execution begins.

If **address** is omitted, execution begins with the instruction pointed to by the current value of the program counter.

Components

Debugger engine.

Example:

```
in>p
```

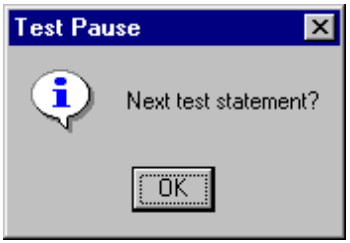
```
A=0x0 HX=0x450 SR=0x70 PC=0xF04E SP=0xFF

00F04E 94               TXS
STEPPED
```

Contents of registers are displayed and the current instruction is disassembled.

PAUSETEST

Displays a modal message box shown in Figure 28.1 on page 712 for testing purpose.

Figure 28.1 Test Pause Message Box



Usage

PAUSETEST

Components

Debugger engine.

Example:

```
in> pausetest
```

PRINTF

The **PRINTF** is the standard ANSI C command: Prints formatted output to the standard output stream.

Usage

PRINTF (“[Text]%format specification” , value)

Components

Debugger engine.

Example

```
in>PRINTF("The value of the counter is: %d", counter)
```

The output is: `The value of the counter is: 2`

PTRARRAY

The **PTRARRAY** command allows to specify if a pointer should be displayed as an array.

Usage

PTRARRY on|off [nb]

Where:

- **on** displays pointers as arrays.
- **off** displays pointers as usual (*pointer).
- **nb** is the number of elements to display in the array when unfolding a pointer displayed as array.

Components

Data component.

Example:

```
in>Ptrarray on 5
```

Display content of pointers as array of 5 items.

RD

The **RD** command displays the content of specified registers. The display of a register includes both the name and hexadecimal representation. If the specified register is not a CPU register, then it looks for this register in a register file as an I/O register. This file is called: **MCUIxxxx.REG** (where **xxxx** is a number related to the MCU).

NOTE This command is processor/derivative specific and will not display banked registers if the processor does not support banking.

Usage

RD { <list> | CPU | * }

where **list** is a list of registers to be displayed. Registers to be displayed are separated by a space. When "**RD CPU**" is specified, all CPU registers are displayed. If no CPU is loaded, "**No CPU loaded**" is displayed as an error message.

When ***** is specified, the **RD** command lists the content of the register file that is currently loaded. If no register file is loaded, following error message is displayed: "**No register file loaded**".

When there is no parameter, the previous **RD** command is processed again. If there is no previous **RD** command, all CPU registers are displayed.

If **list** is omitted, the list and any other parameters of the previous **RD** command are used.

For the first **RD** command of a session, all CPU registers are displayed.

Components

Debugger engine.

Example1:

```
in>rd a hx
```

```
A=0x14
HX=0x2
```

Example2:

```
in>rd cpu
A=0x0 HX=0x450 SR=0x70 PC=0xF04E SP=0xFF
```

RECORD

In the **SoftTrace component**, the **RECORD** command switches frame recording **on** / **off** while the target is running.

Usage

RECORD on|off

Components

SoftTrace component.

Example:

```
in>RECORD on
```

REPEAT

The **REPEAT** command allows you to execute a sequence of commands until a specified condition is true. The **REPEAT** command may be nested.

Press the **Esc** key to stop this command.

Usage

REPEAT

Components

Debugger engine.

Example:

```
DEFINE var = 0
...
REPEAT
  DEFINE var = var + 1
  ...
UNTIL var == 2
```

The REPEAT-UNTIL loop is identical to the ANSI C loop. The operation DEFINE var = var + 1 is done twice, then var = = 2 and the loop ends.

RESET

In the **Profiler and Coverage component**, the **RESET** command resets all recorded frames (statistics).

In the **SoftTrace component**, the **RESET** command resets statistics and recorded frames.

NOTE Make sure that the **RESET** command is redirected to the correct component. Targets also have their own **RESET** command and if **RESET** is not redirected, the target is reset.

Usage

RESET

Components

Profiler and Coverage.

Example:

```
in>Profiler < RESET
```

RESTART

Resets execution to the first line of the current application and executes the application from this point.

Usage

RESTART

Components

Engine component.

Example

```
in>RESTART
```

After the **RESTART** command, the cycle counter is initialized to zero.

RETURN

The **RETURN** command terminates the current command processing level (returns from a CALL on page 661 command). If executed within a command file, control is returned to the caller of the command file (i.e. the first instance that did not chain execution).

Usage

RETURN

Components

Debugger engine.

Example:

In file `d:\demo\cmd1.txt`:

```
...
CALL d:\demo\cmd2.txt
T
...
```

Iin file `d:\demo\cmd2.txt`

```
...
...
RETURN // returns to the caller
```

The command file `cmd1.txt` calls a second command file `cmd2.txt`. It is so necessary to insert the RETURN instruction to return to the caller file. Then the T on page 733 Trace instruction is executed.

RS

The **RS** command assigns new values to specified registers. The **RS** mnemonic is followed by register name and new value(s).

An equal sign (=) may be used to separate the register name from the value to be assigned to the register; otherwise they must be separated by a space. The contents of any number of registers may be set using a single **RS** command. If the specified register is not a CPU register, then the register is searched in a register file as an I/O register. This file is called: `MCUIxxxx.REG` (where **xxxx** is a number related to the MCU).

Usage

RS register[=]value{,register[=]value}

register: Specifies the name of a register to be changed. String register is any of the CPU register names, or name of a register in the register file.

value: is an integer constant expression (in ANSI format representation).

Components

Debugger engine.

Example:

```
in>rs a=0xff hx=0x7fff
```

S

The **S** command stops execution of the emulation processor. Use the Go G on page 689 command to start the emulator.

NOTE The S command ends as soon as the PC is changed.

Usage

S

Alias

STOP

Components

Debugger engine.

Example:

```
in>s
```

```
STOPPING
HALTED
```

Current application debugging is stopped/halted.

SAVE

The **SAVE** command saves a specified block of memory to a specified file in Freescale S-record format. The memory block can be reloaded later using the load S-record (SREC on page 728) command.

NOTE If no path is specified, the destination directory is the current project directory.

Usage

SAVE **range** fileName [offset][;A]

offset: an optional offset to add or subtract from addresses when writing S-records. The default offset is **0**.

;A: appends the saved S-records to the end of an existing file. If this option is omitted, and the file specified by fileName exists, the file is cleared before saving the S-records.

Components

Debugger engine.

Example:

```
in>SAVE 0x1000..0x2000 DUMP.SX ;A
```

The memory range 0x1000..0x2000 is appended to the `DUMP.SX` file.

SAVEBP

The **SAVEBP** command saves all breakpoints of the currently loaded .ABS file into the matching breakpoints file. Also, the matching file has the name of the loaded .ABS file but its extension is .BPT (for example, the Fibo.ABS file has a breakpoint file called FIBO.BPT. This file is generated in the same directory as the .ABS file, when the user quits the Simulator/Debugger or loads another .ABS file.

If on is set, all breakpoints defined in the current application will be stored in the matching .BPT file.

If off is set, all breakpoints defined in the current application will not be stored in the matching .BPT file.

This command is only used in .BPT files and is related to the checkbox **Save & Restore on load** in the Controlpoints Configuration Window. It is used to store currently defined breakpoints (**SAVEBP** on) when the user quits the Simulator/Debugger or loads another .ABS file.

NOTE For more information about this syntax, refer to BS on page 659 command and to the Control Points on page 145 chapter.

Usage

SAVEBP on|off

Components

Debugger engine.

Example:

content of the FIBO.BPT file

```
savebp on
BS &fibo.c:Fibonacci+19 P E; cond = "fibo > 10" E; cdSz = 47 srSz = 0
BS &fibo.c:Fibonacci+31 P E; cdSz = 47 srSz = 0
BS &fibo.c:main+12 P E; cdSz = 42 srSz = 0
BS &fibo.c:main+21 P E; cond = "fiboCount==5" E; cmd = "Assembly < spc
0x800" E; cdSz = 42 srSz = 0
```

SET

Sets a new current target for the debugger by loading the targetName component.

Usage

SET targetName

where targetName is name without extension of the target to set.

Components

Debugger engine.

Example:

```
in>SET Sim
```

The debugger's current target is Simulator.

SETCOLORS

The **SETCOLORS** command is used to change the colors for a specific channel from the Monitor component.

Usage

SETCOLORS ("Name") (Background) (Cursor) (Grid) (Line) (Text

Name is the name of the channel to modify.

Background is the new color for the channel background (the format is : 0x00bbggrr).

Cursor is the new color for the channel cursor (the format is: 0x00bbggrr).

Grid is the new color for the channel grid (the format is: 0x00bbggrr).

Line is the new color for the channel lines (the format is: 0x00bbggrr).

Text is the new color for the channel text (the format is: 0x00bbggrr).

Components

Monitor component.

Example:

```
in>SETCOLORS "Leds.Port_Register bit 0" 0x00123456
0x00234567 0x00345678 0x00456789 0x00567891
```

The color attributes from the channel Leds.Port_Register bit 0 will be changed with these new values.

SLAY

The **SLAY** command is used to save the layout of all window components in the main application window to a specified file.

NOTE Layout files usually have a .**HWL** extension. However, you can specify any file extension.

NOTE If no path is specified, the destination directory is the current project directory.

Usage

SLAY fileName

Components

Debugger engine.

Example:

```
in>slay /hiwave/demo/mylayout.hwl
```

The current debugger layout is saved to the `mylayout.hwl` file in the `/hiwave/demo` directory.

SLINE

With the **SLINE** command, a line of the source file is made visible. If the line is not currently visible, the source will scroll so that it appears on the first line. If the line is currently in a folded part, it is unfolded so that it becomes visible.

NOTE The given line number should be between 1 and number of lines in source file, or else an error message is displayed.

Usage

SLINE line number

Components

Source component

Example:

```
in>sline 15
```

SMEM

In the **Source component**, the **SMEM** command loads the corresponding module's source text, scrolls to the corresponding text location (the code address) and highlights the statements that correspond to this code address range.

In the **Assembly component**, the **SMEM** command scrolls the Assembly component, shows the location (the assembler address) and select/highlights the memory lines of the address range given as the parameter.

In the **Memory component**, the **SMEM** command scrolls the memory dump component, shows the locations (the memory address) of the address range given as the parameter.

Usage

SMEM range

Components

Source, Assembly and Memory components.

Example:

```
in>Memory < SMEM 0x8000,8
```

The Memory component window is scrolled and specified memory addresses are highlighted.

SMOD

In the **Source component**, the **SMOD** command loads/displays the corresponding module's source text. If the module is not found, a message is displayed in Command Line window.

In the **Data component**, the **SMOD** command loads the corresponding module's global variables.

In the **Memory component**, the **SMOD** command scrolls the memory dump component and highlights the first global variable of the module.

NOTE Correct module names are displayed in the Module component window. Make sure that the module name of your command is correct. If the `.abs` is in **HIWARE** format, some debug information is in the object file (`.o`), and module names have a `.o` extension (e.g., `fibo.o`). In **ELF** format, module name extensions are `.c`, `.cpp` or `.dbg` (`.dbg` or program sources in assembler) (e.g., `fibo.c`), since all debugging information is contained in the `.abs` file and object files are not used. Please adapt the following examples with your `.abs` application file format.

Usage

SMOD module

Where **module** is the name of a module taking part of the application. The module name should contain no path. The module extension (i.e. `.DBG` for assembly sources or `.C` for C sources, etc.) must be specified.

The module name is searched in the directories associated with the **GENPATH** environment variable. An error message is displayed:

- If the module specified does not take part of the current application loaded.
- If no application is loaded.

Components

Data, Memory and source components.

Example:

```
in>Data:1 < SMOD fibo.c
```

Global variables found in the `fibo.c` module are displayed in the Data:1 component window.

SPC

In the **Source component**, the **SPC** command loads the corresponding module's source text, scrolls to the corresponding text location (the code address) and highlights the statement that corresponds to this code address.

In the **Assembler component**, the **SPC** command scrolls the Assembly component, shows the location (the assembler address) and select/highlights the assembler instruction of the address given as parameter.

In the **Memory component**, the **SPC** command scrolls the memory dump component, shows the location (the memory address) of the address given as parameter.

Usage

SPC **address**

Components

Assembler, Memory and Source component.

Example:

```
in>Assembly < SPC 0x8000
```

The Assembly component window is scrolled to the address **0x8000** and the associated instruction is highlighted.

SPROC

In the **Data component**, the **SPROC** command shows local variables of the corresponding procedure stack level.

In the **Source component**, the **SPROC** command loads the corresponding module's source text, scrolls to the corresponding procedure and highlights the statement of this procedure that is in the procedure chain.

level = 0 is the current procedure level. **level = 1** is the caller stack level and so on.

NOTE This command is relevant when "C-source" debugging.

NOTE When a procedure of a level greater than 0 is given as parameter to the **SPROC** command, the statement corresponding to the call of the lower procedure is selected.

Usage

SPROC **level**

Components

Data and Source components.

Example:

```
in>Source < SPROC 1
```

This command displays the source code associated with the caller function in the Source component window.

SREC

The **SREC** command initiates the loading of Freescale S-Records from a specified file.

NOTE If no path is specified, the destination directory is the current project directory.

Usage

SREC fileName [offset]

offset: is a signed value added to the load addresses in the file when loading the file contents.

Components

Debugger engine.

Example:

```
in>SREC DUMP.SX
```

The `DUMP.SX` file is loaded into memory.

STEPINTO

The **STEPINTO** command single-steps through instructions in the program, and enters each function call that is encountered.

NOTE This command works while the application is paused in break mode (program is waiting for user input after completing a debugging command).

Usage

STEPINTO

Components

Debugger engine.

Example:

```
in>STEPINTO
```

```
STEP INTO
TRACED
```

TRACED in the status line indicates that the application is stopped by an assembly step function.

STEPOUT

The **STEPOUT** command executes the remaining lines of a function in which the current execution point lies. The next statement displayed is the statement following the procedure call. All of the code is executed between the current and final execution points. Using this command, you can quickly finish executing the current function after determining that a bug is not present in the function.

NOTE This command works while the application is paused in break mode (program is waiting for user input after completing a debugging command).

Usage

STEPOUT

Components

Debugger engine.

Example:

```
in>STEPOUT
```

```
STEP OUT
STARTED
RUNNING
STOPPED
```

STOPPED in the status line indicates that the application is stopped by a step out function.

STEPOVER

The **STEPOVER** command executes the procedure as a unit, and then steps to the next statement in the current procedure. Therefore, the next statement displayed is the next statement in the current procedure regardless of whether the current statement is a call to another procedure.

NOTE This command works while the application is paused in break mode (program is waiting for user input after completing a debugging command).

Usage

STEPOVER

Components

Debugger engine.

Example:

```
in>STEPOVER
```

```
STEP OVER
STARTED
RUNNING
STOPPED
```

STEPPED OVER (or STOPPED) in the status line indicates that the application is stopped by a step over function.

STOP

The **STOP** command stops execution of the emulation processor. Use the Go G on page 689 command to start the emulator.

NOTE The STOP command ends as soon as the PC is changed.

Usage

STOP

Alias

S

Components

Debugger engine.

Example:

```
in>STOP
```

```
STOPPING
HALTED
```

Current application debugging is stopped.

T

The **T** command executes one or more instructions at a specified address, or at the current address (the address in the program counter). The **T** command traces into subroutine calls and software interrupts. For example, if the current instruction is a Branch to Subroutine instruction (**BSR**), the **BSR** is traced, and execution stops at the first instruction of the subroutine. After executing the last (or only) instruction, the **T** command displays the contents of the CPU registers, the instruction bytes at the new address in the program counter and a mnemonic disassembly of the current instruction.

This command can be stopped by typing the **Esc** key.

Usage

T [address][,count]

address: is an address constant expression, the address where execution begins. If **address** is omitted, the instruction pointed to by the current value of the program counter is the first instruction traced.

count: is an integer constant expression, in the decimal integral interval [1, 65535], that specifies the number of instructions to be traced. If **count** is omitted, one instruction is traced.

Components

Debugger engine.

Example:

```
in>T 0xF030
```

```
TRACED
A=0x0 HX=0x7F02 SR=0x62 PC=0xF032 SP=0x44D

00F032 B787             STA    0x87
```

Contents of registers are displayed and current instruction is disassembled.

TESTBOX

Displays a modal message box shown in Figure 28.2 on page 734 with a given string.

Figure 28.2 Test Box Message Box

Usage

TESTBOX "<String>"

Components

Debugger engine.

Example:

```
in>TESTBOX "Step 1: init all vars"
```

TUPDATE

In **Profiler and Coverage components**, the **TUPDATE** command switches the time update feature **on/ off**.

Usage

TUPDATE on|off

Components

Profiler and Coverage components.

Example:

```
in>TUPDATE on
```

UNDEF

The **UNDEF** command removes a symbol definition from the symbol table. This command does not undefine the symbols defined in the loaded application.

Program variables whose names were redefined using the UNDEF on page 735 command are visible again. Undefining an undefined symbol is not considered an error.

Usage

UNDEF symbol | *

If ***** is specified, all symbols defined previously using the command DEFINE are undefined.

Components

Debugger engine.

Example:

```
DEFINE test = 1
...
UNDEF test
```

When the test variable is no longer needed in a command program, it can be undefined and removed from the list of symbols. After UNDEF test, the test variable can no longer be used without (re)defining it.

NOTE See also examples of the DEFINE on page 672 command.

Examples:

The value of an existing symbol can be changed by applying the DEFINE command again. In this case, the previous value is replaced and lost. It is not put on a stack. Then when UNDEF is applied to the symbol, it no longer exists, even if the value of the symbol has been replaced several times:

```
in>DEFINE apple 0
in>LS
```

```
          apple           0x0 (0)     // apple is equal to 0
```

```
in>DEFINE apple = apple + 1
in>LS
```

```
          apple           0x1 (1)     // apple is equal to 1
```

```
in>DEFINE apple = apple + 1
in>LS
```

```
          apple           0x2 (2)     // apple is equal to 2
```

```
in>UNDEF apple
in>LS
```

```
// apple no longer exists
```

In the next example, we assume that the `FIBO.ABS` sample is loaded. At the beginning, no user symbol is defined:

```
in>UNDEF *
in>LS
```

```
User Symbols:    // there is no user symbol
Application Symbols:    // symbols of the loaded application
fiboCount     0x800 (2048)
counter       0x802 (2050)
_startupData  0x84D (2125)
Fibonacci     0x867 (2151)
main          0x896 (2198)
Init          0x810 (2064)
_Startup      0x83D (2109)
```

```
in>DEFINE counter = 1
in>LS
```

```
User Symbols: // there is one user symbol: counter
counter       0x1 (1)
Application Symbols: // symbols of the loaded application
fiboCount     0x800 (2048)
counter       0x802 (2050)
_startupData  0x84D (2125)
Fibonacci     0x867 (2151)
main          0x896 (2198)
Init          0x810 (2064)
_Startup      0x83D (2109)
```

```
in>undef counter
```

```
in>LS
```

```
User Symbols: // there is no user symbol
Application Symbols: // symbols of the loaded application
fiboCount    0x800 (2048)
counter      0x802 (2050)
_startupData 0x84D (2125)
Fibonacci    0x867 (2151)
main         0x896 (2198)
Init         0x810 (2064)
_Startup     0x83D (2109)
```

UNFOLD

In the Source component, the **UNFOLD** command is used to display the contents of folded source text blocks, for example, source text that has been collapsed at program block level. All text is unfolded once or (*****) completely, until no more folded parts are found.

Usage

UNFOLD [*]

Where ***** means unfolding completely, otherwise unfolding only one level.

Components

Source component.

Example:

```
in>UNFOLD *
```

UNTIL

The **UNTIL** keyword is associated with the REPEAT on page 715 command.

Usage

UNTIL condition

Where **condition** is defined as in “C” language definition.

Components

Debugger engine.

Example:

```
repeat
   open assembly
   wait 20
   define i = i + 1
until i==3
```

At the end of the loop, i is equal to 3.

UPDATERATE

Description

In the **Data component** and **Memory component**, the **UPDATERATE** command is used to set the data refresh update rate. This command only has an effect if the Data or Memory component to which it applies is set in Periodical Mode.

Usage

UPDATERATE **rate**

where rate is a constant number matching a quantity of time in tenths of a second, between 1 and 600 tenth of second (0.1 to 60 seconds).

Components

Data and Memory component.

Example:

```
in>Memory < updaterate 30
```

This commands sets the Memory component updaterate to 3 seconds.

VER

The **VER** command displays the version number of the Debugger engine and components currently loaded in the Command line window.

Usage

VER

Components

Debugger engine.

Example:

```
in>ver
```

```
HI-WAVE          6.0.27
HI-WAVE Engine   6.0.49
Source           6.0.20
Assembly         6.0.14
Procedure        6.0.10
Register         6.0.14
Memory           6.0.19
Data             6.0.27
Data             6.0.27
Simulator Target 6.0.17
Command Line     6.0.16
```

In the Command Line component window, Debugger engine and components versions are displayed.

WAIT

The **WAIT** command pauses command file execution for a time in tenths of second or pauses until the target is halted when the option "**;s**" is set.

When no parameter is specified, it pauses for 50 tenths of a second (5 seconds).

When only time is specified, execution of the command file is halted for the specified time.

When only ;s is specified, execution of the command file is halted until the target is halted. If the target is already halted, command file execution is not halted.

When time and ;s are specified:

If the target is running, command file execution is halted for the specified time only if the target is not halted. If the target is halted during the specified period of time (while command file execution is pending), the time delay is ignored and the command file is run.

If the target is already halted, command file execution is not halted (time delay is ignored).

NOTE The Wait instruction ends as soon as the PC is changed.

Usage

WAIT [time] [;s]

Components

Debugger engine.

Example:

```
WAIT 100
T
...
```

Pauses for 10 seconds before executing the T Trace instruction.

WB

The **WB** command sets a specified block of memory to a specified list of byte values. When the range is wider than the list of byte values, the list of byte values is repeated as many times as necessary to fill the memory block. When the range is not an integer, a multiple of the length of the list and the last copy of the list is truncated accordingly. This command is identical to the memory set (MS on page 705) command.

Usage

WB range list

range: is an address range constant that defines the block of memory to be set to the values of the bytes in the list.

list: is a list of byte values to be stored in the block of memory.

Alias

MS

Components

Debugger engine.

Example

```
in>WB 0x0205..0x0220 0xFF
```

This command fills up the memory range 0x0205..0x0220 with the 0xFF byte value.

WHILE

The **WHILE** command allows you to execute a sequence of commands as long as a certain condition is true. The **WHILE** command may be nested.

This command can be stopped by pressing the **Esc** key.

Usage

WHILE condition

Where **condition** is defined as in “C” language definition.

Components

Debugger engine.

Example:

```
DEFINE jump = 0
...
WHILE jump < 20
  DEFINE jump = jump + 1
ENDWHILE
T
...
```

While jump < 100, the jump variable is incremented by the instruction DEFINE jump = jump + 1. Then the loop ends and the T Trace instruction is executed.

WL

The **WL** command sets a specified block of memory to a specified list of longword values. When the range is wider than the list of longword values, the list of longword values is repeated as many times as necessary to fill the memory block. When the range is not an integer or a multiple of the length of the list, the last copy of the list is truncated accordingly.

When a size is specified in the range, this size represents the number of longwords that should be modified.

Usage

WL range list

range: is an address range constant that defines the block of memory to be set to the longword values in the list.

list: is a list of longword values to be stored in the block of memory.

Components

Debugger engine.

Example:

```
in>WL 0x2000 0x0FFFFF0F
```

This command fills up memory starting at address 0x2000 with the 0x0FFFFF0F longword value. The addresses 0x2000 to 0x2003 will be modified.

```
in>WL 0x2000, 2 0x0FFFFF0F
```

This command fills up the memory area 0x2000 to 0x2007 with the longword value 0x0FFFFF0F.

WW

The **WW** command sets a specified block of memory to a specified list of word values. When the range is wider than the list of word values, the list of word values is repeated as many time as necessary to fill the memory block. When the range is not an integer or a multiple of length of the list, the last copy of the list is truncated accordingly.

Usage

WW range list

range: is an address range constant that defines the block of memory to be set to the word values in the list.

list: is a list of word values to be stored in the block of memory.

Components

Debugger engine.

Example:

```
in>WW 0x2000..0x200F 0xAF00
```

This command fills up the memory range 0x2000..0x200F with the 0xAF00 word value.

ZOOM

In the Data component, the **ZOOM** command is used to display the member fields of structures by 'diving' into the structure. In contrast to the UNFOLD on page 737 command, where member fields are not expanded in place. The display of the member fields replaces the previous view. The **ZOOM out** command is used to return to the nesting level indicated by the given identifier.

NOTE Addresses are not needed to zoom out. Simply type "**ZOOM out**".

NOTE This command is relevant when "C-source" debugging.

Usage

ZOOM **address** in|out

Where **address** is the address of the structure or pointer variable that should be zoomed-in or zoomed-out, respectively.

Components

Data component.

Example:

```
in>ZOOM 0x1FE0 in
```

The variable structure located at address **0x1FE0** is zoomed in.

```
in>zoom &_startupData
```

zooms in the **_startupData** structure (**&_startupData** is the address of the **_startupData** structure).

29

Debugger Connection-specific Commands

MMDS Commands

The following paragraphs describe MMDS-specific commands.

Use MMDS-specific commands as you would any other commands, typing them in the Command Line component window, or inserting them into a command file.

For further details about debugger commands, please see the debugger manual section "Debugger Engine Commands" on page 633.

Baud Rate Commands

BAUD: Set Communication Baud Rate

Syntax

`BAUD [rate]`

`rate`: Specifies the new baud rate; must be one of these decimal integer constants:

1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200.

Description

The `BAUD` command sets or displays the baud rate for communication between the system controller and the host computer. For maximum performance, the baud rate should be as high as the host computer can accommodate. The maximum rate is 115200; the default baud rate is 9600.

Without a rate value, the command displays the Communications Baud Rate Specification dialog box for interactive rate selection. If the system does not support the requested rate, the red message, "Error: <baud rate> is not a supported baud rate." appears in the Command Line window.

Example:

```
BAUD 57600
```

Changes the Baud Rate Commands on page 745 to 57600.

Trigger Commands

CT: Clear Triggers

Syntax

CT *list*|*

list: A list of trigger identifiers (A, B, C, or D).

*: All triggers.

Description

The **CT** command clears definition values of specified bus analyzer triggers (events), A, B, C, or D. The command also disables these cleared triggers. Use the Set Trigger (ST) command, or the bus analyzer configuration dialog box, to set trigger values. windows. Triggers are enabled by Use the Trigger Enable (TE) and Trigger Disable (TD) commands to enable and disable triggers.

Example 1:

```
CT A B
```

Clears triggers A and B.

Example 2:

```
CT *
```

Clears all triggers.

This command clears both triggers of a range definition, even though just one of the triggers is a parameter value.

Example 3:

If triggers C and D define a range, `CT C` clears both the triggers.

ST: Set Trigger

Syntax

```
ST [id [[!] [(address | address range | ,) [(data | data
range | ,) [(clips | ,) [LIR= (X | H | L)]]]] [;R | ;W| ;RW]
[;D] ]]
```

Description

The `ST` command defines one of the four bus analyzer triggers. If this command includes only a value or id parameter value, the command interpreter opens the Trigger Specification dialog box. You can use the dialog box to set the trigger value interactively.

`id`: The id of the analyzer trigger: A, B, C, or D.

`!`: The inversion operator for the entire trigger definition. Specifying `!`, as well as an address, data value, and clip value, means that the trigger occurs when the address is not the specified value, or the data value is not the specified value, or the clip value s not the specified value. Specifying `!` with a range is specification means that the trigger occurs at values outside the range, and at the lowest value of the range.

`address`: Address that is to trip the trigger, specified with an address constant:

address[:mask] If you include a mask value, the system comparison involves only the address bits that correspond to ones (1s) of the mask.

`address range`: A range of addresses to trip the trigger. To specify a range, use starting and ending constants, or use a starting address and a length value:

```
start-address[:mask][..end-address]
```

or

```
start-address[:mask][,length]
```

If you include a mask value, the system comparison involves only the address bits that correspond to ones (1s) of the mask. If you include a length and a mask, the system adds the length to the start address, then applies the mask to the start address and to the sum, to obtain the end address.

`,`: The comma directs the analyzer to *not* include an address, address range, data, data range, or clip value in the trigger definition.

data: Data value that is to trip the trigger:

```
value[:mask]
```

If you include a mask value, the system comparison involves only the value bits that correspond to ones (1s) of the mask.

data-range: A range of data values to trip the trigger. To specify a range, use start and end values, or use a starting value and a length:

`start-value[:mask][..end-value]`

or

`start-value[:mask][::length]`

If you include a mask value, the system comparison involves only the value bits that correspond to ones (1s) of the mask. If you include a length and a mask, the system adds the length to the start value, then applies the mask to the start value and to the sum, to obtain the end address.

clips: A 15-bit value that defines MMDS0508 analyzer logic clip signals for the trigger, specified as:

clips[:mask] If you include a mask, the system comparison involves only the value bits that correspond to ones (1s) of the mask. For example, the code `0x1F:0x1F`, sets all clips of Term A to H. Without a mask, the ST command uses the unmasked value `0x1F` for the clips.

Each trigger clip line has three options:

H - High

L - Low

X - Don't Care

The bits of the clips and mask words are:

Bit: Signal:

0 Group A- BRN

1 RED

2 ORG

3 YEL

4 GRN

6 LIR (active low)

`;R`: Trigger on a read bus cycle only.

`;W`: Trigger on a write bus cycle only.

`;RW`: Trigger on a read or write bus cycle.

`;D`: Disable trigger (set trigger value, but disable trigger).

`LIR`: Trigger on a specific value for LIR signal.

`LIR=H` triggers when `LIR` is high.

`LIR=L` triggers when `LIR` is low.

`LIR=X` triggers when `LIR` is high or low.

If either the address or the data value of this command is a range, the analyzer interprets both values as ranges. For example, the analyzer interprets the command `STC 8 20..40` as `STC 8.. 8 20..40`; it interprets the command `STC 8..9 20` as `STC 8..9 20..20`.

If the command lacks any `R`, `W`, or `RW` value, the trigger defaults to read/write.

You can set the LIR signal with the clips (bit 6) or the LIR option.

A bit set in both the clips and masks sets the trigger to high (-H). A bit clear in clips, but set in masks, sets the trigger to low (-L). A bit clear in a mask means that the trigger does not depend on the state of the clip.

Examples:

```
STA 0x1000
```

Sets analyzer trigger A to match accesses at address $1000.

```
STB , 4
```

Sets analyzer trigger B to match accesses with a value of 4, at any address.

```
STC 8 20..40
```

Sets analyzer triggers C and D to match accesses having a value from 20 - 40, at address 8.

```
STC 8..10 20
```

Sets analyzer triggers C and D to match accesses having the value 20, at any address from 8 to 10.

NOTE Mask bits that have the value 0 are “Don’t Care” bits: for those positions of an address, data, or clip value, either a 0 or a 1 trips the trigger. Mask bits set to 1 identify positions whose values must match those of the specified address, data, or clip value. For Address 0xC000, Mask 0xFFFC, loading any of four values in the address bus would trip the trigger: 0xC000, 0xC001, 0xC002, or 0xC003. For Address 0x00B0, Mask 0xFFF0, any address in the range 0x00B0..0x00BF would trip the trigger.

TD: Trigger Disable

Syntax

`TD` *`list`* | `*`

`list`: A list of trigger identifiers (A, B, C, or D), separated by comma or space characters.

`*`: All triggers.

Description

The `TD` command disables specified triggers. Use the Set Trigger (`ST`) command to set triggers; use the Trigger Enable (`TE`) command to enable triggers. Use the Clear Triggers (`CT`) command to clear triggers.

Example:

```
TD A B
```

Disables triggers A and B.

Example:

```
TD *
```

Disables all triggers.

TE: Trigger Enable

Syntax

`TE` *`list`* | `*`

`list`: A list of trigger identifiers (A, B, C, or D), separated by comma or space characters.

`*`: All triggers.

Description

The `TE` command enables specified triggers. Use the Set Trigger (`ST`) command to set triggers; use the Trigger Disable (`TD`) command to disable triggers. Use the Clear Triggers (`CT`) command to clear triggers.

Example 1:

```
TE A B
```

Enables triggers A and B.

Example 2:

```
TE *
```

Enables all triggers.

Bus Analyzer Commands

ARM: Arm Bus Analyser

Syntax

```
ARM
```

Description

The ARM command arms the bus analyzer. When armed, the analyzer records bus cycles while the emulator is executing user code. Arming the analyzer clears the current contents of the trace buffer. Use the disarm analyzer (DARM) command to disarm the analyzer.

DARM: Disarm Bus Analyser

Syntax

```
DARM
```

Description

The DARM command disarms the bus analyzer. When disarmed, the analyzer does not record bus cycles. If the bus analyzer is disarmed already, this command does nothing. ARM arms the analyzer

GE: Go To Event

Syntax

`GE` *`list`* | `*` [`;B`]

`;B`: Specifies a backward search; without this option, the command directs a forward search.

`list`: A list of events (`A`, `B`, `C`, or `D`), separated by space or comma characters.

Description

The `GE` command searches in the analyzer trace buffer for a frame that matches the specified event. A forward search begins with the frame immediately following the current frame; a backward search begins at the frame immediately preceding the current frame.

Example:

```
GE A B
```

Moves the cursor to the next frame which contains the event A and/or B.

GF: Go To Frame

Syntax

```
GF frame
```

`frame`: Frame number, in the range 1..8191. This number must be decimal number, regardless of the current default number base.

Description

The GF command moves the cursor to a specified trace buffer frame. If the command's frame-number value is greater than the number of frames stored in the buffer, the command moves your display to the buffer's last frame.

Example 1:

```
GF 4096
```

Moves the cursor to frame 4096.

Example 2:

```
GF 8191
```

Moves the cursor to the last frame in the buffer.

GP: Go to Analyzer Search Pattern

Syntax

```
GP [;B]
```

;B: Specifies a backward search; without this option, the command directs a forward search.

Description

The `GP` command searches the analyzer trace buffer for a frame that matches the defined search pattern. (Use the `SP` command to define the pattern.) A forward search begins at the frame immediately following the current frame. A backward search begins at the frame immediately preceding the current frame.

When the search finds a matching frame, the system positions the line cursor on the frame, in the center of the screen. If no frame matches the pattern, the line cursor does not move. If no search pattern is defined, this command searches for any pattern: that is, the line cursor moves to the next line (or preceding line).

Example 1:

```
GP
```

Searches forward for the next frame that matches the search pattern.

Example 2:

```
GP ;B
```

Searches backward for the previous frame that matches the search pattern.

LT: Log Trace

Syntax

```
LT [range]
```

range: The range of frames (decimal numbers) to be copied. Without a range value, this command copies all frames in the trace buffer.

Description

The LT command copies contents of the bus analyzer trace buffer to the log file, in the current view format. The target must be stopped to execute an LT command.

Example 1:

```
LF logfile
```

Open the logging file.

Example 2:

```
LT
```

Copy all the trace buffer to the logging file.

Example 3:

```
NOLF
```

Close the logging file.

SC: Set Analyzer Clock

Syntax

`SC [timetag [frequency]]`

`timetag`: One of these values:

`OSC1MHZ` (1 Mhz oscillator)

`OSC2MHZ` (2 Mhz oscillator)

`OSC4MHZ` (4 Mhz oscillator)

`OSC8MHZ` (8 Mhz oscillator)

`OSC16MHZ` (16 Mhz oscillator)

`EXT` (external clock)

`BUS` (bus clock)

`PROGRAM` (programmable clock)

`frequency`: Time tag clock frequency for the programmable clock, 50 Hz to 50 kHz (enter a decimal hertz value). The programmable clock source has a fundamental frequency of 500,000 Hz; the actual frequency the system uses is the quotient of 500,000 Hz divided by an integer. Thus, 50000 is valid, but 49000 is not. (If you specify 49000, the system rounds the value up the the next higher valid frequency: 50000.)

Description

The `SC` command sets the source for the time tag clock, which increments the trace buffer time tag. If this command includes no parameter values, the command interpreter opens the Bus Analyzer Configuration dialog box. You can use the Time Tag Clock page to reach the Clock Specification dialog box, then set the clock sources interactively.

Example 1:

```
SC PROGRAM 100
```

Sets the programmable clock, running at 100Hz, as the time clock source.

Example 2:

```
SC OSC8MHZ
```

Sets the 8 Mhz oscillator as the time tag clock source.

SP: Set Analyzer Search Pattern

Description

The SP command defines a search pattern for the bus analyzer trace buffer. The go to pattern (GP) command starts the search for the pattern defined by the most recent SP command. If the SP command includes no parameter values, the command interpreter opens the Bus Analyzer Configuration dialog box. You can use the Pattern page to define the pattern interactively.

Syntax

```
SP [!] [(<address> | ,) [(<data> | ,) [(<clips> | ,)[LIR =
(X | L  | H)]]]][;R | ;W| ;RW]
```

`!:`

The inversion operator, which applies to a single address or data value. Specifying `!` sets a pattern on addresses or data values *other* than the specified address or data value.

`address:`

Address part of the pattern, specified with an address constant:

```
address [:mask]
```

If you include a mask value, the system comparison involves only the address bits that correspond to ones (1s) of the mask.

`,:`

The comma directs the analyzer to *not* include an address, data, or clips value in the pattern definition. The omitted item matches any value of its type (so is a don't care entry).

`data:`

Data part of the pattern, specified as:

```
value [:mask]
```

If you include a mask value, the system comparison involves only the value bits that correspond to ones (1s) of the mask.

A 15-bit value that defines MMDS0508 analyzer logic clip signals for the trigger, specified as:

`clips[:mask]`

`clips:`

A 16-bit value that defines the logic clip signal values of the pattern, specified as:

`clips [:mask]`

If you include a mask, the system comparison involves only the value bits that correspond to ones (1s) of the mask.

The bits of the clips and mask words are:

Table 29.1 Clips and Mask Work Bit Assignments

Bit		Signal		Bit		Signal
0	Group A -	BRN		8	Group B -	BRN
1		RED		9		RED
2		ORG		10		ORG
3		YEL		11		YEL
4		GRN		12		GRN
5		BLU		13		BLU
6		LIR		14		VLT
7		GRY		15		GRY

`;R`: Search for a read bus cycle only.

`;W` : Search for a write bus cycle only.

`;RW`: Search for either a read or write bus cycle.

`LIR`: Search for a specific `LIR`-signal value:

`LIR=H` searches for `LIR` high.

`LIR=H` searches for `LIR` low.

`LIR=X` searches for `LIR` either high or low.

A bit set in both the clips and masks detects the pattern if the bit is high (-H). A bit is clear in clips but set in masks detects the pattern if the bit is low (-L). A bit clear in a mask means that pattern detection does not depend on the state of the clip.

Examples:

```
SP 0x1000
```

Sets accesses at address $1000 to be part of the analyzer search pattern.

```
SP , 4
```

Sets accesses with the data value 4, at any address, to be part of the analyzer search pattern.

```
SP 8 20
```

Sets accesses with the data value 20, at address 8, to be part of the analyzer search pattern.

NOTE Mask bits that have the value 0 are “Don’t Care” bits: for those positions of an address, data, or clips value, either a 0 or a 1 trips the trigger. Mask bits set to 1 identify positions whose values must match those of the specified address, data, or clip value. For Address 0xC000, Mask 0xFFFC, loading any of four values in the address bus would trip the trigger: 0xC000, 0xC001, 0xC002, or 0xC003. For Address 0x00B0, Mask 0xFFF0, any address in the range 0x00B0..0x00BF would trip the trigger.

SQ: Set Sequencer

Syntax

`SQ [mode [count] [;S] ]`

`mode`: One of these values:

`ALL` (records all bus cycles)

`EVENT` (records only events)

`SEQ0` (sequential A+B+C+D)

`SEQ1` (sequential A+B->C+D)

`SEQ2` (sequential A->B->C D<-)

`SEQ3` (sequential A->B->C->D)

`SEQ4` (nth event after A+B+C+D)

`count`: Terminal count for the `ALL` and `EVENT` modes (a decimal value). Directs recording of count frames (of the appropriate type). When the number of frames recorded reaches the count value, the analyzer stops recording and is disarmed. If this command has no count value, the analyzer records continuously whenever the emulator is running. For sequential modes, the count value is the decimal post-trigger count, directing recording of count bus cycles after the trigger point. The default count value is 1. If no count value follows the mode value `ALL`, the system records frames for all cycles. If no count value follows the mode value `EVENT`, the system records all the events.

`;S`: Specifies stopping the emulator when recording stops, letting you use the analyzer as a sequenced breakpoint machine.

Description

The `SQ` command sets the analyzer sequencer mode. If this command includes no parameter values, the command interpreter opens the Bus Analyzer Configuration dialog box: select the Sequencer Setup dialog box to program the sequencer interactively. Selecting from the Setup menu is another way to open the Sequencer Setup dialog box.

Example 1:

```
SQ ALL 100
```

Specifies 100 bus cycles. After 100 cycles, the analyzer disarms itself and stops recording.

Example 2:

```
SQ EVENT 10 ;S
```

Specifies recording 10 event cycles, then stopping the emulator.

TT: Display Time Tag Difference

Syntax

```
TT [sframe [eframe]]
```

sframe: The starting frame number.

eframe: The ending frame number; without this value, the system uses the end frame in the trace buffer to calculate the difference.

Description

The TT command displays the difference between the time tags in the two trace frames of the analyzer display. Without frame-number values, this command displays the difference between the beginning and ending frames

Example 1:

```
TT
```

Displays the difference between the beginning and ending frames in the trace buffer.

Example 2:

```
TT 80
```

Displays the difference between frame 80 and the ending frame in the trace buffer.

Example 3:

```
TT 10 40
```

Displays the difference between frames 10 and 40 in the trace buffer.

VA: Analyzer View

Syntax

```
VA [MODE=(MIX | INS | GRAPH)]
```

MODE=: When followed by one of the three following keywords, specifies the mode of the bus analyzer window display.

MIX: Mixed view mode.

INS: Instruction view mode.

GRAPH: Graphical view mode.

Description

The VA command specifies the display format for the analyzer trace buffer. This command is valid only if the bus analyzer is open.

Example:

```
VA MODE=MIX
```

Selects mixed view mode.

Target Signal Emulation Commands

SIG: Set Emulator Signals

Syntax

`SIG [ [ENABLE] signal {signal}] [ DISABLE signal {signal}]`

`signal`: The signal to be enabled or disabled: `RESETIN` or `RESETOUT`.

`ENABLE`: Connect the signal from the target system.

`DISABLE`: Disconnect the signal from the target system.

Description

The `SIG` command enables or disables control signals from the target MCU. If the command specifies any signals, but lacks either keyword `ENABLE` nor `DISABLE`, the command enables the signals.

Example:

```
SIG ENABLE RESETIN
```

Enables the “reset in” signal from the target system.

Reset Commands

RESET: Reset Target MCU

Syntax

RESET [**GO**|**STOP**]

GO: Resets the MCU and does a Go from Reset.

STOP: Resets the MCU and stops (default).

Description

The **RESET** command resets the target MCU.

Examples:

```
Reset Go
```

Resets the MCU and does a GO from Reset.

```
Reset
```

Resets the MCU and stops (default setting).

Other Commands

LOADMAP: Load Memory Map

Syntax

`LOADMAP fileName | MCUID`

`fileName`: a standard full path of a file that defines a memory map.

`MCUID`: the MCU identifier. If the command includes an MCUID value, the system loads a memory map from the file that matches this value.

Description

The `LOADMAP` command loads a memory map from a file. The file specification must be the full pathname. If the file is in the current directory, the characters "`.\`" must precede the name (for example: `LOADMAP .\00123V22.MEM`).

Example:

```
LOADMAP 0xC17
```

Loads a memory map from a file that matches this MCU identifier (68HC08AX48).

Note that the MCUID identifies an MCU, not an EM Board. The memory map filename has this format:

`0nnnnVvv.MEM`

where 'nnnn' is the four-digit, hexadecimal 'MCU-ID', and 'vv' is a two-digit version number.

MEM: Display Memory Map

Syntax

`MEM`

Description

The `MEM` command displays the current memory map.

Example:

```
in>MEM
Type    Addresses   Comment
-------------------------------------------------------
IO         0..  3F  PRU or TOP   TOP board resource or the PRU
NONE      40..  4F  NONE
RAM       50.. 64F  RAM
NONE     650.. 7FF  NONE
EEPROM   800.. A7F  EEPROM
NONE     A80..3DFF  NONE
ROM     3E00..FDFF  ROM
IO      FE00..FE1F  PRU or TOP   TOP board resource or the PRU
NONE    FE20..FFDB  NONE
ROM     FFDC..FFFE  ROM
COP     FFFF..FFFF  special ram for cop
RT MEM     0.. 3FF  (disabled)
-------------------------------------------------------
```

OSC: Select Emulator Clock Frequency

Syntax

```
OSC [rate | source]
```

Description

The OSC command selects the emulator clock frequency: one of five internally generated frequencies (16 Mhz, 8 Mhz, 4 Mhz, 2 Mhz, or 1 Mhz), or an external clock source. The firmware sets the default emulator clock rate, adapting it to the current frequency.

Entering this command without parameters opens the corresponding dialog box.

```
OSC [rate | source]
```

where *rate* is:

OSC1MHZ	Selects the 1 Mhz oscillator.
OSC2MHZ	Selects the 2 Mhz oscillator.
OSC4MHZ	Selects the 4 Mhz oscillator.
OSC8MHZ	Selects the 8 Mhz oscillator.
OSC16MHZ	Selects the 16 Mhz oscillator.

Where *source* is:

EXT	Selects an external clock source.

Example:

```
OSC OSC8MHZ     // Use the 8 Mhz internal emulator clock.
```

PROTOCOL: Controls Show Protocol Functionality

Syntax

```
PROTOCOL [ON | OFF]
```

ON: Reports commands and responses in the command line window (default value for this command).

OFF: Does not report commands or responses in the command line window; does not log commands or responses in the log file.

Description

The PROTOCOL command controls reporting commands and responses in the command line window. Entering this command without a parameter value has the same effect as entering the command with the parameter value ON.

RTMEM: Configure Real Time Memory

Syntax

`RTMEM` *`address`* `[;E|;D]`

`address`: base address of a real time memory block (default is 0).

`;E`: Enables the real time memory block (the default for this command).

`;D`: Disables the real time memory block.

Description

The `RTMEM` command lets you enable or disable real time memory.

Example:

```
RTMEM 0x1000 ;D
```

Disables the real time memory block at base address $1000.

MMEVS Commands

This appendix explains the commands specific to the MMEVS.

Use these commands as you would any others, typing them in the Command Line component or when inserting them into a command file.

For further details about commands, please see the *Debugger* manual appendix, *Debugger Commands*, as well as the section *Command Line Component*.

Baud Rate Commands

BAUD: Set Communication Baud Rate

Syntax

```
BAUD [rate]
```

`rate`: specifies the new baud rate, which must be one of these decimal integer constants:

```
1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200
```

Description

The `BAUD` command sets or displays the baud rate for communication between the system controller and the host computer. For maximum performance, the baud rate should be as high as the host computer can accommodate. The maximum rate is 115200; the default baud rate is 9600.

Without a rate value, the command displays the Communications Baud Rate Specification dialog box for interactive rate selection. If the system does not support the requested rate, the red message, “Error: <baud rate> is not a supported baud rate.”appears in the Command Line window.

Example:

```
BAUD 57600
```

Changes the communication baud rate to 57600.

Target Signal Emulation Commands

SIG; Set Emulator Signals

Syntax

`SIG [ [ENABLE] signal {signal}] [ DISABLE signal {signal}]`

`signal`: The signal to be enabled or disabled: `RESETIN` or `RESETOUT`.

`ENABLE`: Connect the signal from the target system.

`DISABLE`: Disconnect the signal from the target system.

Description

The `SIG` command enables or disables reset control signals from the target MCU. If the command specifies any signals, but lacks either keyword `ENABLE` or `DISABLE`, the command enables the signals.

Examples:

```
SIG ENABLE RESETIN
```

Enables the “reset in” signal from the target system.

Reset Commands

RESET: Reset Target MCU

Syntax

`RESET [GO|STOP]`

`GO`: Resets the MCU and does a Go from Reset.

`STOP`: Resets the MCU and stops (default).

Description

The `RESET` command resets the target MCU.

Examples:

```
Reset Go
```

Resets the MCU and does a `GO` from Reset.

```
Reset
```

Resets the MCU and stops (default setting).

After the system asserts the reset is asserted, it executes the `RESET.CMD` command file.

Other Commands

LOADMAP: Load Memory Map

Syntax

`LOADMAP fileName | MCUID`

`fileName`: the full pathname of a file that defines a memory map.

`MCUID`: the MCU identifier. If the command includes an MCUID value, the system loads a memory map from the file that matches this value.

Description

The `LOADMAP` command loads a memory map from a file. The file specification must be the full pathname. If the file is in the current directory, the characters " `.\` " must preceded the name (for example: `LOADMAP .\00123V22.MEM`).

Example:

`LOADMAP 0xC17`

Loads a memory map from a file that matches this MCU identifier (for the 68HC08AX48 MCU).

The MCUID identifies an MCU, not an EM Board. The memory map filename has this format:

0nnnnVvv.MEM

where `nnnn` is the four-digit, hexadecimal `MCU-ID`, and `vv` is a two-digit version number.

MEM: Display Memory Map

Syntax

`MEM`

Description

The `MEM` command displays the current memory map.

Example:

```
in>MEM
Type    Addresses   Comment
-------------------------------------------------------
IO         0..  3F  PRU or TOP   TOP board resource or the PRU
NONE      40..  4F  NONE
RAM       50.. 64F  RAM
NONE     650.. 7FF  NONE
EEPROM   800.. A7F  EEPROM
NONE     A80..3DFF  NONE
ROM     3E00..FDFF  ROM
IO      FE00..FE1F  PRU or TOP   TOP board resource or the PRU
NONE    FE20..FFDB  NONE
ROM     FFDC..FFFE  ROM
COP     FFFF..FFFF  special ram for cop
RT MEM     0.. 3FF  (disabled)
-------------------------------------------------------
```

OSC: Select Emulator Clock Frequency

Syntax

```
OSC [rate | source]
```

Description

The OSC command selects the emulator clock frequency: one of five internally generated frequencies (16 Mhz, 8 Mhz, 4 Mhz, 2 Mhz, or 1 Mhz) or an external clock source. The firmware sets the default emulator clock rate, adapting it to the current frequency.

Entering this command without parameters opens the corresponding dialog box.

```
OSC [rate | source]
```

where rate is:	OSC1MHZ	Selects the 1 Mhz oscillator.
	OSC2MHZ	Selects the 2 Mhz oscillator.
	OSC4MHZ	Selects the 4 Mhz oscillator.
	OSC8MHZ	Selects the 8 Mhz oscillator.
	OSC16MHZ	Selects the 16 Mhz oscillator.
Where source is:	EXT	Selects an external clock source.

Example:

```
OSC OSC8MHZ     // Use the 8 Mhz internal emulator clock.
```

PROTOCOL: Controls Show Protocol Functionality

Syntax

```
PROTOCOL [ON | OFF]
```

ON: Reports commands and responses in the command line window (default value for this command).

OFF: Does not report commands or responses in the command line window; does not log commands or responses in the log file.

Description

The PROTOCOL command controls reporting commands and responses in the command line window. Entering this command without a parameter value has the same effect as entering the command with the parameter value ON.

Book VI - Environment Variables

Book VI Contents

Each section of the Debugger manual includes information to help you become more familiar with the Debugger, to use all its functions and help you understand how to use the environment. This book, the Debugger Environment Variables, defines the HC08 and HC(S)08 environment variables, both those environment variables used by the debugger engine and those specific to individual debugger connections.

This book is divided into the following chapters:

- Debugger Engine Environment Variables on page 777
- Connection-specific Environment Variables on page 799

30

Debugger Engine Environment Variables

This chapter describes the environment variables that the Debugger uses. Other tools, such as the Linker, also use some of these environment variables. For more information about other tools, see their respective manuals.

Click any of the following links to jump to the corresponding section of this chapter:

- Debugger Environment on page 778
- Local Configuration File (usually project.ini) on page 781
- ABSPATH: Absolute Path on page 788
- DEFAULTDIR: Default Current Directory on page 789
- ENVIRONMENT=: Environment File Specification on page 790
- GENPATH: #include “File” Path on page 791
- LIBRARYPATH: ‘include <File>’ Path on page 792
- OBJPATH: Object File Path on page 793
- TMP: Temporary directory on page 794
- USELIBPATH: Using LIBPATH Environment Variable on page 795
- Search Order for Source Files on page 796
- Debugger Files on page 796

Debugger Environment

Various parameters of the Debugger may be set using environment variables. The syntax is always the same:

```
Parameter = KeyName "=" ParamDef.
```

NOTE Do not use blanks in the definition of an environment variable.

For example:

```
GENPATH=C:\INSTALL\LIB;D:\PROJECTS\TESTS;/usr/local/lib;/
home/me/my_project
```

The Debugger parameters may be defined in several ways:

- Using system environment variables supported by your operating system.
- Putting the definitions in a file called `DEFAULT.ENV` in the default directory.

NOTE The maximum length of environment variable entries in the `DEFAULT.ENV/.hidefaults` is 4096 characters.

- Putting definitions in a file given by the value of the system environment variable ENVIRONMENT.

NOTE The default directory mentioned above can be set by using the system environment variable DEFAULTDIR: Default Current Directory on page 789.

When looking for an environment variable, all programs first search the system environment, then the `DEFAULT.ENV` file and finally the global environment file given by ENVIRONMENT. If no definition can be found, a default value is assumed.

NOTE Ensure that no spaces exist at the end of environment variables.

The Current Directory

The most important environment for all tools is the current directory. The current directory is the base search directory where the tool begins to search for files (for example, the `DEFAULT.ENV / .hidefaults` file)

Normally, the current directory of a tool is determined by the operating system or program that launches another one (for example, WinEdit).

For MS Windows-based operating systems, the current directory definition is more complex.

- If the tool is launched using a File Manager/Explorer, the current directory is the location of the executable launched.
- If the tool is launched using an Icon on the Desktop, the current directory is the one specified and associated with the Icon.
- If the tool is launched by dragging a file on the icon of the executable under Windows 95, 98, Windows NT 4.0 or Windows 2000, the desktop is the current directory.
- If the tool is launched by another tool with its own current directory specified (for example, WinEdit), the current directory is the one specified by the launching tool (for example, current directory definition in WinEdit).
- For the Debugger tools, the current directory is the directory containing the local project file. Changing the current project file also changes the current directory, if the other project file is in a different directory. Note that browsing for a C file does not change the current directory.

To overwrite this behavior, the environment variable DEFAULTDIR: Default Current Directory on page 789 may be used.

Global Initialization File (MCUTOOLS.INI - PC Only)

All tools may store global data in `MCUTOOLS.INI`. The tool first searches for this file in the directory of the tool itself (path of executable). If there is no MCUTOOLS.INI file in this directory, the tool looks for the file in the MS Windows installation directory (for example, `C:\WINDOWS`).

Example:

```
C:\WINDOWS\MCUTOOLS.INI
D:\INSTALL\PROG\MCUTOOLS.INI
```

If a tool is started in the `D:\INSTALL\PROG\DIRECTORY`, the project file in the same directory as the tool is used (`D:\INSTALL\PROG\MCUTOOLS.INI`).

If the tool is started outside the `D:\INSTALL\PROG` directory, the project file in the Windows directory is used (`C:\WINDOWS\MCUTOOLS.INI`).

NOTE For more information about `MCUTOOLS.INI` entries, see the compiler manual.

Local Configuration File (usually project.ini)

The Debugger does not change the `default.env` file. Its content is read only. All configuration properties are stored in the configuration file. The same configuration file can be used by different applications.

The shell uses the configuration file with the name “project.ini” in the current directory only. That is why this name is also suggested to be used with the Debugger. Only when the shell uses the same file as the compiler, the editor configuration written and maintained by the shell can be used by the Debugger. Apart from this, the Debugger can use any file name for the project file. The configuration file has the same format as windows `.ini` files. The Debugger stores its own entries with the same section name as in the global `mcutools.ini` file.

The current directory is always the directory containing the configuration file. If a configuration file in a different directory is loaded, then the current directory also changes. When the current directory changes, the `default.env` file is reloaded. Always when a configuration file is loaded or stored, options in the environment variable COMPOPTIONS are reloaded and added to the project options. Beware of this behavior when a different `default.env` file exists in different directories, which contain incompatible options in COMPOPTIONS.

When a project is loaded using the first `default.env`, its COMPOPTIONS are added to the configuration file. If this configuration is stored in a different directory, where a `default.env` file exists with incompatible options, the Debugger adds options and marks the inconsistency. Then a message box appears to inform the user that the default.env options were not added. In such a situation the user can either remove the option from the configuration file with the option settings dialog or remove the option from default.env with the shell or a text editor, depending on which options should be used in the future.

At startup there are three ways to load a configuration:

- use the command line option **prod**
- the project.ini file in the current directory
- or **Open Project** entry from the file menu.

If the option **prod** is used, then the current directory is the directory the project file is in. If **prod** is used with a directory, the project.ini file in this directory is loaded.

Default Layout Configuration (PROJECT.INI)

The default layout activated when starting the Debugger is defined in the `PROJECT.INI` file located in the project directory, as shown in Listing 30.1 on page 782. All default layout related parameters are stored in section **[DEFAULTS]**.

Listing 30.1 Example Content of PROJECT.INI:

```
[HI-WAVE]
Window0=Source      0    0  60  30
Window1=Assembly   60    0  40  30
Window2=Procedure   0   30  50  15
Window3=Terminal    0   45  50  15
Window4=Register   50   30  50  30
Window5=Memory     50   60  50  30
Window6=Data        0   60  50  15
Window7=Data        0   75  50  15
Target=Sim
```

Target: Specifies the target used when starting the Debugger (loads the file **<target>** with a **.tgt** extension), for example, Target=Sim for HC(S)12(X) Freescale Full Chip Simulator, or Target=Motosil, Target=Bdi.

Window<n>: Specifies coordinates of the windows that must be open when the Debugger is started. The syntax for a window is:

Window<n>=<component> <XPos> <YPos> <width> <height>

where **n** is the index of the window. This index is incremented for each window and determines the sequence windows are opened. This index is relevant in case of overlapping windows, because it determines which window will be on top of the other. Values for the index have to be in the range **0..99**.

component specifies the type of component that should be opened, for example, **Source**, **Assembly**, etc.

XPos specifies the X coordinate of the top left corner of the component (in percentage relative to the width of the main application client window).

YPos specifies the Y coordinate of the top left corner of the component (in percentage relative to the height of the main application client window).

width specifies the width of the component (in percentage relative to the width of the main application client window).

height specifies the height of the component (in percentage relative to the height of the main application client window).

Example:

Window5=Memory 50 60 50 30

Window number 5 is a Memory component, its starting position is at: 50% from main window width, 60% from main window height. Its width is 50% from main window width and its height 30% from main window height.

Other Parameters

- It is possible to load a previously saved layout from a file by inserting the following line in your `PROJECT.INI` file:

Layout=<LayoutName>

Where **LayoutName** is the name of the file describing the layout to be loaded,for example, **Layout=lay1.hwl**

NOTE The layout path can be specified if the layout is not in the project directory.

NOTE If **Layout** is defined in `PROJECT.INI`, the **Layout** parameter overwrites any **Window<n>** definition, describing the default windows layout.

- It is possible to load a previously saved project from a file by inserting the following line in your `PROJECT.INI` file:

Project=<ProjectName>

where **ProjectName** is the name of the file describing the project to be loaded,

for example, **Project=Proj1.hwc**

NOTE The project path can be specified if the project is not in the project directory. This option can be used for compatibility with the old .hwp format (Project=oldProject.hwp) and will be opened as a new project file.

See File Menu on page 14 section for more details about Projects.

NOTE If **Layout** and **Project** are defined in `PROJECT.INI`, the **Project** parameter overwrites the **Layout** parameter, also containing layout information.

MainFrame=<nbr.>,<nbr.>,<nbr.>,<nbr.>,<nbr.>,<nbr.>,

<nbr.>,<nbr.>,<nbr.>,<nbr.>

This variable is used to save and load the Debugger main window states: positions, size, maximized, minimized, iconized when opened, etc. This entry is used for internal purposes only.

- The toolbar, status bar, heading line, title bar and small border can be specified in the default section:

The toolbar can be shown or hidden with the following syntax:

Tooolbar = (0 | 1)

If 1 is specified, the toolbar is shown, otherwise the toolbar is hidden.

The status bar can be shown or hidden with the following syntax:

Statusbar = (0 | 1)

If 1 is specified, the status bar is shown, otherwise the toolbar is hidden.

Title bars can be shown or hidden with the following syntax:

Hidetitle = (0 | 1)

If 1 is specified, the title bars are hidden, otherwise they are shown.

The heading lines can be shown or hidden with the following syntax:

Hideheadlines = (0 | 1)

If 1 is specified, the heading lines are hidden otherwise they are shown.

The border can be reduced with the following syntax:

Smallborder = (0 | 1)

If 1 is specified, borders are thin otherwise they are normal.

- The environment variable BPTFILE authorizes the creation of breakpoint files; they may be enabled or disabled. All breakpoints of the currently loaded 'abs' file are saved in a breakpoints file. BPTFILE may be ON (default) or OFF. When ON, breakpoint files are created. When OFF, breakpoint files are not created.

BPTFILE =(On | Off)

NOTE Target specific environment variables can also be defined in the `PROJECT.INI` file. Refer to the specific target manual for details.

Ini File Activation

When a project file (`PROJECT.INI`) is activated, the following occurs (from first action to last):

- The old Project file is closed.
- Target Component is unloaded
- The environment variable (Path) is added from the Project file.

Select HI-WAVE section to retrieve value from:

- if an entry 'Windows0' or 'Target' can be retrieved from section [HI-WAVE] then:

  use [HI-WAVE]
- else if an entry 'Windows0' or 'Target' can be retrieved from section [DEFAULTS] then:

use [DEFAULTS]

- else:

  use [HI-WAVE]

The environment variables are loaded from the default.env file.

If an entry ’Layout=lll’ exists, the layout file lll.hwl is loaded and executed.

The target is set (if entry 'Target=ttt' exists load target 'ttt').

If an entry ’Project=ppp’ exists, the command file ’ppp’ is executed.

The configuration file (*.hwc) is loaded (entry configuration=*.hwc).

Environment Variable Paths

Most environment variables contain path lists indicating where to search for files. A path list is a list of directory names separated by semicolons following the syntax below:

PathList = DirSpec {";" DirSpec}.

DirSpec = ["*"] DirectoryName.

Example:

```
GENPATH=C:\INSTALL\LIB;D:\PROJECTS\TESTS;/usr/local/hiwave/
lib;/home/me/my_project
```

If a directory name is preceded by an asterisk ("*"), the programs recursively search the directory tree for a file, not just the given directory. Directories are searched in the order they appear in the path list.

Example:

```
GENPATH=.\;*S;O
```

NOTE Some DOS environment variables (like GENPATH, LIBPATH, etc.) are used.

We strongly recommend working with WinEdit and setting the environment by means of a DEFAULT.ENV file in your project directory. This 'project directory' can be set in WinEdit's 'Project Configure...' menu command. This way, you can have different projects in different directories, each with its own environment.

NOTE When using WinEdit, do **not** set the system environment variable `Defaultdir`. If you do and this variable does not contain the project directory given in WinEdit’s project configuration, files might not be put where you expect them.

Line Continuation

It is possible to specify an environment variable in an environment file (default.env/ .hidefaults) over multiple lines by using the line continuation character '\':

Example:

```
OPTIONS=\
-W2 \
-Wpd
```

This is the same as:

```
OPTIONS=-W2 -Wpd
```

Be careful when using the line continuation character with paths. For example:

```
GENPATH=.\
TEXTFILE=.\txt
```

Will result in:

```
GENPATH=.TEXTFILE=.\txt
```

To avoid such problems, use a semicolon';' at the end of a path, if there is a '\' at the end:

```
GENPATH=.\;
TEXTFILE=.\txt
```

Environment Variables

The remainder of this section is devoted to describing each of the environment variables available for the Debugger. The options are listed in alphabetical order and each is divided into several sections described in the following table, Environment Variable Details on page 787.

Table 30.1 Environment Variable Details

Topic	Description
Tools	Lists of other tools that are using this variable
Synonym	Fore some environment variables a synonym also exists. The synonyms may be used for older releases of the Debugger and will be removed in the future. A synonym has lower precedence than the environment variable.
Syntax	Specifies the syntax of the option in EBNF format.
Arguments	Describes and lists optional and required arguments for the variable.
Default	Shows the default setting for the variable or none.
Description	Provides a detailed description of the option and how to use it.
Example	Gives an example of usage and effects of the variable where possible. The examples show an entry in the **default.env** file for PC.
See also	Names related sections.

ABSPATH: Absolute Path

Tools

SmartLinker, Debugger

Synonym

None

Syntax

```
ABSPATH=" {<path>}.
```

Arguments

<path>: Paths separated by semicolons, without spaces.

Description

When this environment variable is defined, the SmartLinker will store the absolute files it produces in the first directory specified. If ABSPATH is not set, the generated absolute files will be stored in the directory the parameter file was found.

Example:

```
ABSPATH=\sources\bin;..\..\headers;\usr\local\bin
```

DEFAULTDIR: Default Current Directory

Tools

Compiler, Assembler, Linker, Decoder, Librarian, Maker, Burner, Debugger.

Synonym

None.

Syntax

```
"DEFAULTDIR=" <directory>.
```

Arguments

<directory>: Directory specified as default current directory.

Default

None.

Description

With this environment variable the default directory for all tools may be specified. All tools indicated above will take the directory specified as their current directory instead of the one defined by the operating system or launching tool (for example, editor).

NOTE This is an environment variable at the system level (global environment variable). It CANNOT be specified in a default environment file (DEFAULT.ENV/.hidefaults).

Example:

```
DEFAULTDIR=C:\INSTALL\PROJECT
```

See also:

The Current Directory on page 779
Global Initialization File (MCUTOOLS.INI - PC Only) on page 780

ENVIRONMENT=: Environment File Specification

Tools

Compiler, Linker, Decoder, Librarian, Maker, Burner, Debugger.

Synonym

HIENVIRONMENT

Syntax

```
"ENVIRONMENT=" <file>.
```

Arguments

<file>: file name with path specification, without spaces

Default

None.

Description

This variable has to be specified at the system level. Normally the application looks in the The Current Directory on page 779 for an environment file named `default.env`. Using ENVIRONMENT (for example, set in the autoexec.bat for DOS), a different file name may be specified.

NOTE This is an environment variable at the system level (global environment variable). It CANNOT be specified in a default environment file (DEFAULT.ENV/.hidefaults).

Example:

```
ENVIRONMENT=\Freescale\prog\global.env
```

GENPATH: #include “File” Path

Tools

Compiler, Linker, Decoder, Burner, Debugger.

Synonym

HIPATH

Syntax

```
"GENPATH=" {<path>}.
```

Arguments

<path>: Paths separated by semicolons, without spaces.

Default

Current directory

Description

If a header file is included with double quotes, the Debugger searches in the current directory, then in the directories given by GENPATH and finally in the directories given by LIBRARYPATH: ‘include <File>’ Path on page 792.

NOTE If a directory specification in this environment variable starts with an asterisk (“*”), the whole directory tree is searched recursively. All subdirectories and their subdirectories are searched. Within one level in the tree, search order is random.

Example:

```
GENPATH=\sources\include;..\..\headers;
\usr\local\lib
```

See also:

Environment variable LIBPATH

LIBRARYPATH: ‘include <File>’ Path

Tools

Compiler, ELF tools (Burner, Linker, Decoder)

Synonym

LIBPATH

Syntax

```
"LIBRARYPATH=" {<path>}.
```

Arguments

<path>: Paths separated by semicolons, without spaces.

Default

Current directory

Description

If a header file is included with double quotes, the Compiler searches in the current directory, then in the directories given by GENPATH: #include “File” Path on page 791 and finally in directories given by LIBRARYPATH: ‘include <File>’ Path on page 792.

NOTE If a directory specification in the environment variables starts with an asterisk (“*”), the whole directory tree is searched recursively. All subdirectories and their subdirectories are searched. Within one level in the tree, search order is random.

Example:

```
LIBRARYPATH=\sources\include;..\..\headers;\usr\local\lib
```

See also:

Environment variable GENPATH: #include “File” Path on page 791

Environment variable USELIBPATH: Using LIBPATH Environment Variable on page 795

OBJPATH: Object File Path

Tools

Compiler, Linker, Decoder, Burner, Debugger.

Synonym

None.

Syntax

```
"OBJPATH=" <path>.
```

Default

Current directory

Arguments

<path>: Path without spaces.

Description

If a tool looks for an object file (for example, the Linker), then it first checks for an object file specified by this environment variable, then in GENPATH: #include “File” Path on page 791 and finally in HIPATH.

Example:

```
OBJPATH=\sources\obj
```

TMP: Temporary directory

Tools

Compiler, Assembler, Linker, Librarian, Debugger.

Synonym

None.

Syntax

```
"TMP=" <directory>.
```

Arguments

<directory>: Directory to be used for temporary files.

Default

None.

Description

If a temporary file has to be created, normally the ANSI function tmpnam() is used. This library function stores the temporary files created in the directory specified by this environment variable. If the variable is empty or does not exist, the current directory is used. Check this variable if you get an error message “Cannot create temporary file”.

NOTE This is an environment variable at the system level (global environment variable). It CANNOT be specified in a default environment file (DEFAULT.ENV/.hidefaults).

Example:

```
TMP=C:\TEMP
```

See also:

The Current Directory on page 779

USELIBPATH: Using LIBPATH Environment Variable

Tools

Compiler, Linker, Debugger.

Synonym

None.

Syntax

```
"USELIBPATH=" ("OFF" | "ON" | "NO" | "YES")
```

Arguments

"ON", "YES": The environment variable LIBRARYPATH: ‘include <File>’ Path on page 792 is used to look for system header files <*.h>.
"NO", "OFF": The environment variable LIBRARYPATH: ‘include <File>’ Path on page 792 is not used.

Default

ON

Description

This environment variable allows a flexible usage of the LIBRARYPATH: ‘include <File>’ Path on page 792 environment variable, because LIBRARYPATH: ‘include <File>’ Path on page 792 may be used by other software (for example, version management PVCS).

Example:

```
USELIBPATH=ON
```

See also:

Environment variable LIBRARYPATH: ‘include <File>’ Path on page 792

Search Order for Source Files

This section describes the search order (from first to last) used by the debugger.

In the Debugger for C Source Files (*.c, *.cpp)

1. Path coded in the absolute file (.abs)
2. Project file directory (where the .pjt or .ini file is located)
3. Paths defined in the GENPATH environment variable (from left to right)
4. Abs File directory

In the Debugger for Assembly Source Files (*.dbg)

1. Path coded in the absolute file (.abs)
2. Project file directory (where .pjt or .ini file is located)
3. Paths defined in the GENPATH environment variable (from left to right)
4. Abs File directory

In the Debugger for Object Files (HILOADER)

1. Path coded in the absolute file (.abs)
2. Abs File directory
3. Project file directory (where .pjt or .ini file is located)
4. Path defined in the OBJPATH environment variable
5. Paths defined in the GENPATH environment variable (from left to right)

Debugger Files

The Debugger comes with several program, application, configuration files and examples. These files and file extensions are listed in the following table.

Table 30.2 Debugger File Extensions

File Extension	Description
*.ABS	Absolute framework application file e.g., fibo.abs
*.ASM	Assembler specific file e.g., macrodem.asm
*.BBL	Burner Batch Language file e.g, fibo.bbl
*.BPT	Debugger Breakpoint file e.g., fibo.bpt
*.C *.CPP	C and C++ source files
*.CHM	Compiled HTML help file
*.CMD	Command File Script, for example, Reset.cmd
*.CNF	Specific cpu configuration file
*.CNT	Help Contents File, for example, cxa.cnt
*.CPU	Central Processor Unit Awareness file
*.DBG	Debug listing files, for example, Fibo.dbg
DEFAULT.ENV	Debugger Default Environment file.
*.DLL	A .DLL file that contains one or more functions compiled, linked, and stored separately from the processes that use them. The operating system maps the DLLs into the process's address space when the process is starting up or while it is running. The process then executes functions in the DLL. The DLL of the Debugger is provided for supported library and extended functions.
*.H	Header file
HIWAVE.EXE	The Debugger for Windows executable program.
*.HWL	Debugger Layout file, for example, default.hwl
*.HWC	Debugger Configuration file (project.hwc)
*.EXE	Other Windows executable program, for example, LINKER.EXE
*.FPP	Flash Programming Parameters files (CPU specific) for example, mcu0e36.fpp
*.HLP	Application Help file, for example, Hiwave.hlp

Table 30.2 Debugger File Extensions

File Extension	Description
*.IO	I/O's simulation file, for example, sample11.io
*.ISU	Uninstall Application File
*.PJT	Debugger configuration Settings File, for example, Project.pjt
*.INI	Debugger configuration Settings File, for example, Project.ini
*.LST	Assembler Listing File, for example, fibo.lst
*.MCP	Freescale CodeWarrior IDE project file
*.MAK	Make file, for example, demo.mak
*.MAP	Mapping file, for example, macrodem.map
*.MEM	Memory Configuration file, for example, 000p4v01.mem
*.MON	Firmware loading, file for allowing to load a specified target, for example, Firm0508.mon
*.O	Object file code, for example, Fibo.o
*.PDF	Portable Document Format file.
*.PRM	Linker parameter file, for example, `fibo.prm`
Project.Ini	Debugger Project Initialization File
*.REC	Recorder File
*.REG	Register Entries files, for example, mcu081e.reg
*.SIM	CPU Awareness file, for example, st7.sim
*.SX	S-Record file, for example, fibo.sx
*.TXT	General Text Information file.
*:TGT	Target File for the Debugger, for example, xtend-g3.tgt
*.WND	Debugger Window Component File, for example,, recorder.wnd
*.XPR	Debugger User Expression file, for example, Fibo.xpr

31

Connection-specific Environment Variables

Some of the environment variables that can be used in the debugging process are imported with the connection software and are specific to that connection. This chapter is intended to list and describe those variables.

Connection-specific Environment Variables

There are currently no connection environment variables that can be manually edited.

Book VII - Appendixes

Book VII Contents

This book is divided into the following chapters:

- Debugger Legacy on page 803

A

Debugger Legacy

Legacy PEDebug Target Interface

The PEDebug Target Interface has become “Legacy” and has been replaced by specific “Debugger Connections”. The “Target Interface” term has been replaced by the term “Connection”.

New P&E Connections for HC08/HCS08

For the HC08 CPU, the following P&E connections have been created to replace the former “Target Interfaces”:

- Full Chip Simulation Connection
- Mon08 Interface Connection
- ICS Mon08 Interface Connection
- P&E Mutlilink/Cyclone Pro Connection
- ICS P&E Mutlilink/Cyclone Pro Connection

For the HCS08 CPU, the P&E following connections have been created:

- Full Chip Simulation Connection
- P&E Mutlilink/Cyclone Pro Connection

Please refer to specific manual chapter to get further details about each connection.

“Revert to Full Chip Simulator” Feature Removed

These new connections will NO LONGER revert to default Full Chip Simulator when the hardware cannot be contacted. It is now necessary to open the Set Connection dialog box and to choose the Full Chip Simulator if that is the connection you want. The Full Chip Simulator is now a Connection with the same priority as any other connection.

Connection Selection or Change Always Available within IDE

The CW08v5.x IDE features a new menu option in the Project menu. The Change MCU/ Connection option opens the Device and Connection dialog box, where you can change the currently selected project connection.

Automatic Upgrade Path for Projects Previously Created

The debugger will automatically upgrade the previous connection (target) to the current latest connection set in a previous project. Please find further details in the Technical Note **TN 241**.

Legacy Target Interfaces Removed

The following hardware Target Interfaces have been discontinued:

For HC08 CPU:

- Hitex Emulator Target Interface
- MON08 Target Interface (originally ìHIWAREî MON08 Target Interface)
- Trace32 Target Interface (Lauterbach Emulator)

For HCS08 CPU:

- BDM_HCS08 Target Interface

HC08 and HCS08 Full Chip Simulator Components NO LONGER Supported

The following components can be opened in the debugger layout within the Open Window Component dialog box, or with the OPEN command in the Command window. However, these components are only operational with the Legacy “Freescale HC(S)08 Instruction Set Simulator”, (former **Sim.tgt** target interface). Therefore, HC08 and HCS08 Full Chip Simulators do NOT support the following debugger components, which remain selectable and available for backward compatibility purposes only:

- Adc_dac
- Io_led
- Io_ports

- IT_keyboard
- Lcd
- Led
- Monitor
- Phone
- Push_buttons
- Segments_display
- Softtrace
- Stimulation
- Taillight
- Template
- Testterm
- Timer
- Wagon
- Winlift

Index

Symbols

.abs file 45
.cmd 55
.hidefaults 779, 789, 790, 794
.hwl 783
.HWP 15
.hwp 783
.INI 15
.MEM 451
.PJT 15
.rec 100
.WND 45
.wnd 35
.xpr file 66

A

A 570, 644
About 552, 628
About Box 33
About True Time Simulator and Real Time Debugger 33
ABSPATH 788
ACTIVATE 644
Add New Instrument 131, 132
Additional MON08 Connection Menu Options 339
Address 78, 81, 594
Address B 586, 587
Address... 49
ADDXPR 645
Align 133
All Text Folded At Loading 120
Analog 134
AND Mask 137, 138, 140
Application
    Assembly Step 197
    Embedded 5
    Loading 193
    Starting 194
    Step In 195
    Step Out 196
    Step Over 196
    Stopping 194
    Target 5
ARM 752
Arming 465
arming trace 428
Arrange Icons 32
ASCII 81
Assembly Step 23
Assembly Step Out 24
Assembly Step Over 24
AT 655
ATTRIBUTES 645
Auto 104
Automatic 67, 80
Automatically analyze the FIFO content 596
AUTOSIZE 655

B

B 570
Background Color 30
Backgroundcolor 134, 135
Bar 134
Barcolor 136
Bardirection 136
BASE 656
BAUD 745, 769
BAUDRATE 455, 456, 493, 499, 500
BC 656
BCKCOLOR 657
BD 658
Bin 69, 79, 104, 198
Binary 198, 201
Bit Reverse 80, 104
Bitnumber to Display 139

Bottom 133
Bounding Box 135
Breakpoint 48, 110
    Checking condition 149
    Command 158
    Conditional 156, 174
    Counting 155, 174
    Definition 145
    Deleting 157
    Multiple selection 149
    Permanent 145
    Position 152
    Temporary 145, 153
breakpoint 568
Breakpoint with Register Condition 156, 157
Breakpoints... 24
BS 659
Bus Analyzer 456
Bus State Analyzer
    arming 428
    cables 406
    defining events 424
    selecting record mode 426
    viewing data 429
Bus Trace 552, 563, 568
Byte 79

C

-C 11
Cables, Connecting
    Host Computer 437, 485
    Power 438, 486
    Target 438, 486
CALL 661
Call Chain 94
Capture 590
Capture r/write values at Address B after access at Address A 590
Capture the read/write values at Address B 590
Cascade 32
CD 662
CF 663
Clear 602
CLOCK 665
clock speed
    time tag 428
Clone Attributes 133
CLOSE 665
-Cmd 11
CMDFILE 666
Code coverage 583
Code profiling 583
CodeWarrior Integration 207
collecint data 428
Collecting Data 464
Color if 140
Color if Bit 139
COM1 447
COMDEV 455, 493, 499, 500
Command 143
    Syntax 633
Command File Dialog 27
Command File menu entry 27
Command File Playing 55
Command Line 9
Commands 745, 769
communication
    USB 410
Communication Baud Rate 449, 494
Communication Configuration 444, 489
COMPLEMENT
    DATA Component 650
    Memory Component 651
    Register Component 647
complex breakpoint 568
Component
    Assembly 48, 193, 194
    Associated Menus 34
    Command Line 53
    Coverage 57
    CPU 45
    DAC 62
    Data 64, 193, 194, 197
    Framework 6
    Inspector 122
    Main Menu 34
    Memory 76, 203
    MicroC 88

Module 92
Pop Up Menu 35
Profiler 96
Recorder 100
Register 103, 194, 201
SoftTrace 106
Source 109, 193, 194
Target 46
Terminal 205
VisualizationTool 129
Window 45
components 405
Components File 35
COMPOPTIONS 781
Configuration 16, 444, 489
Connection (HC08FCS) Dropdown Menu 250
Connector, Cable
Pin Assignments 487
Signal Descriptions 487
Continuous 461
Control Point
Definition 145
Dialogs 145
Controlpoints Configuration 572, 573, 578
Copy 132
COPYMEM 666
CopyMem 78
Copyright 33
Counted 462
Coverage 583, 604
CPU
Cycle 13
cycle 103
CR 667
Cross-debugging 5
CT 746
Ctrl+E 131
Ctrl+L 132
Ctrl+S 132
CTRL-P 134
Current Directory 779, 789
Customize 19
Cut 133
CYCLE 667
Cycle 107

D

DAC
Configure the file types 214
Configuring 211
Configuring the tools 219
Database directory 213
Debugger Interface 222
Debugger name 228
IDE 211
library path 214
Ndapi.dll 227
new project 212
Preprocessor | Header Directories 215
Preprocessor | Preinclude file 216
Project root directory 213
Referential project root directory 213
Requirements 211
rue Time Simulator and Real Time Debugger project file 225
Source 215
Synchronized debugging 227
Troubleshooting 227
User help file 213
working directories 212
DARM 752
DASM 668
Data 574
data
viewing trace 429
Data Dump 476
Data Format 448, 493
Data value 603
DB 669
DBG 598
DBG FIFO Data 602
DBGCA 575, 577, 582
DBGCB 577, 582
DBGFH 596, 602
DBGFL 596, 602, 603
DBGT 583
DDE
HI-WAVE server 209

DDEPROTOCOL 671
Debugger Start Option -C 11
Debugger Start Option -Cmd 11
Debugger Start Option -ENVpath 11
Debugger Start Option -Instance=%currentTargetName 10
Debugger Start Option -Nodefaults 10
Debugger Start Option -Prod 10
Debugger Start Option -T 9
Debugger Start Option -Target 10
Debugger Start Option -W 10
Debugger Trigger Register 583
Debugger, starting 8
debuggers
    ICD08SW
        limitations 377
    tips 378
Debugging 5
Dec 69, 79, 104, 198
Decimal 198
Decimalmode 140
default IP address 413
DEFAULT.ENV 778, 779, 789, 790, 794
default.env 493
DEFAULTDIR 789
DefaultDir 190
DEFINE 672
Delete Breakpoint 51, 114
Delete Trigger 594
Delete Trigger Address 571
DETAILS 673
Disable Breakpoint 51, 114
Disabled mode 584
Disarm automatically the module when the debugger stops 596
Disarming 465
Display 78
    Graphical 468
    Instructions Only 468
    Textual 466
Display Absolute Address 49
Display Adress 49
Display Adress Dialog 81
Display Code 49
Display Headline 134
Display Scrollbars 134
Display Symbolic 49
Display Version 140
Displayfont 142
DL 674
Do not halt when the fifo is full 592
documentation 405
Dragging 36, 38
Driving True Time Simulator and Real Time Debugger trough DDE 209
Dual-Port RAM 453
DUMP 675
Dump 601
Dumping 476
DW 675
dynamic 578

E

E 676
e 576
Editing
    Memory 203
    Register 201
    Variable 199
Editmode 131, 132, 134
Editor 65
ELSE 677
ELSEIF 677
EM
    Installing 436, 485
    Removing 437, 485
emulation module (EM)
    defined 406
Enable Breakpoint 51, 114
ENDFOCUS 678
ENDFOR 679
ENDIF 679
ENDWHILE 680
Environment
    ABSPATH 788
    DEFAULTDIR 789
    ENVIRONMENT 778
    File 778

GENPATH 791, 793
HIENVIRONMENT 790
HIPATH 791, 793
LIBPATH 792, 795
LIBRARYPATH 792
OBJPATH 793
TMP 794
USELIBPATH 795
Variable 787
-ENVpath 11
EQUAL Mask 137, 140
Erase Flash 399, 562
events
defining 424
record modes 426
EXECUTE 680
EXIT 681
Exit 16
Expert 582
Expert mode 575
Expert triggers 582
Explorer 779
Expression Command File 66
Expression Editor 65

F

FCS - ADC Module Option 253
FCS - ADC User Commands 255
FCS - Clock Generation Module Commands 258
FCS - Clock Generation Module Option 256
FCS - External Interrupt Module 264
FCS - High Resolution PWM (HRP) module 258
FCS - High Resolution PWM Commands 260
FCS - HRP Commands 260
FCS - Input/Output (I/O) Ports Module 262
FCS - Input/Output Ports Commands 263
FCS - IRQ Commands 265
FCS - Keyboard Interrupt Commands 267
FCS - Keyboard Interrupt Module 266
FCS - MSCAN Controller Module 273
FCS - Multi-Master Inter-Integrated Circuit (MMIIC) Module 268
FCS - Multi-Master Inter-Integrated Circuit Module Commands 270
FCS - Run Till Cycle Command 251
FCS - USB Commands 324
FCS Configuration Procedure 249
FCS Connection Menu - ADC Module Option 253
FCS Connection Menu - Clock Generation Module Option 256
FCS Connection Menu - Device Option 250
FCS Connection Menu - External Interrupt Module 264
FCS Connection Menu - High Resolution PWM Module 258
FCS Connection Menu - Input/Output (I/O) Ports Module 262
FCS Connection Menu - Keyboard Interrupt Module 266
FCS Connection Menu - Modules (Peripherals) Options 253
FCS Connection Menu - Multi Master Inter-Integrated Circuit Module Option 268
FCS Connection Menu - Programmable Timer Interrupt Module 287
FCS Connection Menu - Select Device Option 250
FCS Connection Menu - Serial Communications Interface Module 288
FCS Connection Menu - Serial Peripheral Interface Module Option 296
FCS Connection Menu - Slave LIN Interface Controller Module 291
FCS Connection Menu - Timer Interface Module Option 300
FCS Connection Menu - Universal Serial Bus (USB) Module 305
FCS Connection Menu - View Register Files Option 252
FCS Peripherals (Modules) Options 253
features 404
Field Description 142, 143
FIFO 596, 598, 602
FIFO Analyse remark 598
FIFO Depth 602, 603
File
Environment 778

File Manager 779
Filename 137
FILL 681
Fill Memory Dialog 81
FILTER 682
FIND 683
Find 116, 118
Find Procedure 116, 119
FINDPROC 684
FLEXlm 6
Float 104
FOCUS 685
FOLD 686
Fold 120
Fold All Text 120
Folding 112
    Mark 112
Folding Menu 119
Foldings 116
FONT 686
Fonts 30
FOR 687, 700
Format 78, 198, 201
Format mode 142
Format... 67
FPRINTF 688
Frame 598
FRAMES 688
Frames 106
Frozen 67, 70, 80
FSICE - Assigning an IP Address 413
FSICE - Bus Analyzer Configuration Dialog Box 425
FSICE - Bus State Analyzer (BSA) 423
FSICE - Communication Dialog Box 409
FSICE - Communication Through Ethernet Port 408
FSICE - Communication Through USB Port 410
FSICE - Emulation System Reset 421
FSICE - Establishing Communication 408
FSICE - Logic Cables And Connectors 422
FSICE - Memory Map Dialog Box 418
FSICE - Overview 403
FSICE - Setting Up The Hardware 407
FSICE - Setting Up The System 410
FSICE - Setup System 406
FSICE - Specifying a Memory Map 416
FSICE - Specifying Comm Information 412
FSICE - Specifying the Clock Speed 419
FSICE - System Components 405
FSICE - System Requirements 403
FSICE - Target Signals Dialog Box 420
FSICE Basae
    IP address 413
FSICEBASE
    components 405
    features 404
    requirements 403
FSICEBASE Emulator 403
Full Chip Simulation (FCS) Connection 249
Full Chip Simulation Module Commands 251
Fuse Replacement 442

G

G 689
GDI 588, 597
GE 753
GENPATH 791, 793
GF 753
Global 68
Global Variable
    Displaying 197
GO 690
Go to Frame 601
Go To Line 119
Go to Line 115, 117, 118
GP 754
Graphic bar 58, 96
Graphical 599
Graphical display 599
GRAPHICS 693
Grid Color 134
Grid Mode 134
Grid Size 134

H

Halt 23
Halt when the fifo is full 592

Hardware 5
hardware
components 405
setup 407
hardware breakpoints 581
Hardware Connection 489
Hardware Installation
Configuring the Platform Board 435, 483
Connecting Cables 437, 485
Fuse Replacement 442
Installing the EM 436, 485
Introduction 433
Pin Assignments, Connector 440, 487
Removing the EM 437, 485
Reset Switch 439, 487
Signal Descriptions, Connector 440, 487
HC08 Full Chip Simulation 249
HC08FCS Menu - ADC Module Extended Menu Options 253
HCS08 567
HCS08 FCS - ADC Module 507
HCS08 FCS - ADC Module Commands 509
HCS08 FCS - Clock Generation Module 509
HCS08 FCS - Clock Generation Module Commands 511
HCS08 FCS - Configuration Procedure 503
HCS08 FCS - Connection (HCS08FCS) Menu 504
HCS08 FCS - Connection Menu Module Commands 505
HCS08 FCS - External Interrupt (IRQ) Module 518
HCS08 FCS - HCS08FCS Menu Device Option Exteded Menus 505
HCS08 FCS - Input/Output (I/O) Ports Module 516
HCS08 FCS - Input/Output Ports User Commands 517
HCS08 FCS - Inter-Integrated Circuit Module 512
HCS08 FCS - Inter-Integrated Circuit Module Commands 513
HCS08 FCS - IRQ Commands 519
HCS08 FCS - Keyboard Interrupt Commands 522
HCS08 FCS - Keyboard Interrupt Module 520
HCS08 FCS - Modulo Timer Interrupt Module 523
HCS08 FCS - Peripheral Modules Commands 507
HCS08 FCS - SCI Commands 525
HCS08 FCS - Serial Communications Interface Module 524
HCS08 FCS - Serial Peripheral Interface Module 527
HCS08 FCS - Set Connection Dialog Box 503
HCS08 FCS - SPI Commands 529
HCS08 FCS - Timer Interface Module 532
HCS08 FCS - Timer Module Commands 534
HCS08 Full Chip Simulation 503
HCS08 Multilink/Cyclone Pro - Additional Connection Menu Options 542
HCS08 Multilink/Cyclone Pro - Advanced Programming/Debug Option 542
HCS08 Multilink/Cyclone Pro - Bus Trace Option 545
HCS08 Multilink/Cyclone Pro - Connection Assistant Dialog Box 538
HCS08 Multilink/Cyclone Pro - Connection Procedure 537
HCS08 Multilink/Cyclone Pro - Device Option 540
HCS08 Multilink/Cyclone Pro - Device Option Extended Menu 541
HCS08 Multilink/Cyclone Pro - Expert Programmer Window 544
HCS08 Multilink/Cyclone Pro - Set Connection Dialog Box 537
HCS08 Multilink/Cyclone Pro - Set Connections Dialog Box 537
HCS08 Multilink/Cyclone Pro - Start Expert Mode Programmer Option 543
HCS08 Multilink/Cyclone Pro - Trigger Module Settings Option 545
HCS08 Multilink/Cyclone Pro - View Register Files Option 544

HCS08 MultilinkCyclonePro - Connection Menu 540
HCS08 P&E Multilink/Cyclone PRO Connection 537
HCS08 Serial Monitor 567, 588
HCS08 Serial Monitor - CodeWarrior and Serial Monitor Connection 555
HCS08 Serial Monitor - Connection Menu Options 562
HCS08 Serial Monitor - Debugger Main Window Component Menu 558
HCS08 Serial Monitor - Derivative Selection Dialog Box 559, 565
HCS08 Serial Monitor - First Steps From Within an Existing Project 558
HCS08 Serial Monitor - First Steps Using the Stationery Wizard 556
HCS08 Serial Monitor - Monitor Communication Tab 563
HCS08 Serial Monitor - Monitor Setup Window 559
HCS08 Serial Monitor - Monitor Setup Window Monitor Communication Tab 563
HCS08 Serial Monitor - Monitor Setup Window Vector Table Mirroring Tab 561
HCS08 Serial Monitor - Set Connection Dialog Box 558
HCS08 Serial Monitor - Technical Considerations 555
HCS08 Serial Monitor - Vector Table Mirroring Tab 564
HCS08 Serial Monitor Connection 555
HCS08 SerialMonitor - HC(S)08 New Project Wizard Window 556
Height 134
HELP 693
Help Topics 33
Hex 68, 69, 79, 104, 198, 201
Hexadecimal 198, 201, 204
Hide Headline 19
Hide Tile 19
HIENVIRONMENT 790
High Display Value 136, 138, 143
HIPATH 791
Horiz. Text Alignment 142
Horizontal Size 133
How To ... 189

I

ICD08SW
    MON08
        interface. See MON08
        limitations 377
        tips 378
ICS MON08 - Added Connection Menu Options 354
ICS MON08 - Advanced Options Dialog Box 355
ICS MON08 - Advanced Programming/Debug Options 354
ICS MON08 - Connection Manager Window - Advanced Settings Tab 348
ICS MON08 - Connection Manager Window - Connect to Target Tab 347
ICS MON08 - Connection Manager Window STATUS Area 352
ICS MON08 - Connection Menu Advanced Programming/Debug Options 354
ICS MON08 - Connection Menu Start Expert Mode Programmer Option 355
ICS MON08 - Connection Menu View Register Files Option 356
ICS MON08 - Cycle Power Button 350
ICS MON08 - Expert Mode Programmer Window 356
ICS MON08 - Interface Selection Dialog Box 347
ICS MON08 - Power Up and Power Down Buttons 349
ICS MON08 - Start Expert Mode Programmer Option 355
ICS MON08 - Target MCU Security Bytes 351
ICS MON08 - Tpd And Tpu Timing Textboxes 348
ICS MON08 - View Register Files Option 356
ICS MON08 Cable Connection Communications Type 349
ICS MON08 Connection Procedure 345

ICS MON08 Interface Connection 345
ICS MON08 Set Connection Dialog Box 346
ICS Multilink/Cyclone Pro - Additional Connection Menu Options 391
ICS Multilink/Cyclone Pro - Advanced Options Dialog Box 391
ICS Multilink/Cyclone Pro - Advanced Programming/Debug Option 391
ICS Multilink/Cyclone Pro - Advanced Settings Tab 385
ICS Multilink/Cyclone Pro - Connection Manager Window - Advanced Settings Tab 385
ICS Multilink/Cyclone Pro - Connection Manager Window Connect to Target Tab 383
ICS Multilink/Cyclone Pro - COnnection Manager Window STATUS Area 388
ICS Multilink/Cyclone Pro - Connection Menu Added Menu Options 391
ICS Multilink/Cyclone Pro - Connection Procedure 382
ICS Multilink/Cyclone Pro - Cycle Power Up/ Down Button 387
ICS Multilink/Cyclone Pro - Device Class Description 394
ICS Multilink/Cyclone Pro - Expert Mode Programmer Window 392
ICS Multilink/Cyclone Pro - MON08 Cable Connection Communications Type 386
ICS Multilink/Cyclone Pro - Power Up/Down Buttons 386
ICS Multilink/Cyclone Pro - Remove a Manually Configured Interface 384
ICS Multilink/Cyclone Pro - Set Connection Dialog Box 382
ICS Multilink/Cyclone Pro - Start Expert Mode Programmer Option 392
ICS Multilink/Cyclone Pro - Target MCU Security Bytes 388
ICS Multilink/Cyclone Pro - Tpd And Tpu Timing Textboxes 385
ICS Multilink/Cyclone Pro - View Register Files Option 393
ICS P&E Multilink/Cyclone PRO Connections 381
ICS08RKW
  Register Block Window 252, 341, 356, 376, 393, 506, 544
IDF 209
IF 694, 700
I-LOGIX 88
inDART-HC08 > About 399
inDART-HC08 > MCU Configuration 399
inDART-HC08 > User’s Manual 399
inDART-HCS08 567
Indicatorcolor 136, 139
Indicatorlength 136
init.cmd 192
INSPECTORUPDATE 695
-Instance=%currentTargetName 10
INSTRUCTION 577
Instruction 578
Instruction at Address A and value on data bus match 588
Instruction at Address A and value on data bus mismatch 589
Instruction at Address A is executed 587
Instruction at Address A or Address B is executed 587
Instruction at Address A then at Address B were executed 588
Instruction execution inside Address A - Address B range" 588
Instruction execution outside Address A - Address B range" 588
Instruction Syntax 634
Instructions display 598
Interfacing
  Hardware Connection 444
IP address
  default 413
IPATH 793
Items 600

J

Jumper Headers
  Factory Test (J1) 483
  Port Voltage Control (J2–J4) 435, 484

K

keyword DAC
True Time Simulator and Real Time Debugger project file 225
keyword Hardware Connection 444
Kind of Port 135

L

Layout 7, 783
Layout - Load/Store 32
Left 133
LF 696
LIBPATH 795
LIBRARYPATH 792
Line Continuation 786
LOAD 697
Load Application 15
Load Layout 132
Load Target 25
LOADCODE 698
Loading an Application 193
LOADMAP 453, 497, 765
LOADSYMBOLS 698
Local 68
Local Variable
Displaying 197
Locked 67, 70
LOG 699
Low Display Value 136, 138, 143
LS 703
LT 755
Lword 79

M

MainFrame 783
Markpoints 573, 578
markpoints 571
Marks 116
match value 586, 588
MCU Clock 454, 498
MCU clock 454, 498
MCU Communication 399, 552, 628
MCU-Id 451, 495
MCUTOOLS.INI 190, 780
MEM 704, 766, 773
Memory 453, 498, 574
Dump 76
Word 76
Memory access at Address A 585
Memory access at Address A and value on data bus match 585
Memory access at Address A and value on data bus mismatch 586
Memory access at Address A or at Address B 585
Memory access at Address A then memory access at Address B 585
Memory access inside Address A - Address B range 585
Memory Configuration 450, 495
Memory Map 450
memory map
modifying 417
viewing 416
Memory Write Access 595
Menu 448
Help 33
Run 22
Target 24, 29
View 19
Window 31
MicroC 88
mismatch value 586, 589
MMDS - Cable Connections 437
MMDS - Communication Configuration 446
MMDS - Communication Device Specification 447
MMDS - Communication Option 449
MMDS - Component Menu Set Connection Option 445
MMDS - Components 432
MMDS - Configuring the Platform Board 435
MMDS - Connection 443
MMDS - Connector Information 439
MMDS - Data Format 448
MMDS - Debugger Status Bar 448
MMDS - Default Connection Setup 455
MMDS - General Description 431

MMDS - Installation 433
MMDS - Installing the EM 436
MMDS - Interfacing Your System and the Connection 444
MMDS - Loading MMDS Connection 446
MMDS - Loading MotoSIL Connection 446
MMDS - Loading the MMDS0508 Connection 445
MMDS - Memory Configuration Dialog Box 451
MMDS - Memory Map Option 450
MMDS - Open Personality File 452
MMDS - Power Supply Fuse Replacement 442
MMDS - Removing the EM 437
MMDS - Signals Emulation Option 453
MMDS - Target Signals Dialog Box 454
MMDS0508 444
MMDS0508 Emulator 431
MMEVS - Cable Connections 485
MMEVS - Communication Configuration 491
MMEVS - Communication Device Specification 492, 494
MMEVS - Component Menu Set Connection Option 490
MMEVS - Components 482
MMEVS - Configuring the Platform Board 483
MMEVS - Data Format 493
MMEVS - Debugger Status Bar 493
MMEVS - Default Connection Setup 499
MMEVS - General Description 481
MMEVS - Installation 482
MMEVS - Installing the EM 485
MMEVS - Interfacing Your System and Connection 489
MMEVS - Loading the MMEVS0508 Connection 490
MMEVS - Memory 498
MMEVS - Memory Configuration 495
MMEVS - MMEVS Connection 488
MMEVS - MMEVS0508 Connection Menu 493
MMEVS - Personality Files 495
MMEVS - Removing the EM 485
MMEVS - Signals Emulation 498
MMEVS0508 488, 489
Mode 78
Modes
    counted 461
    non-triggered 461
    triggered 462
Modify Trigger 593
Module 92
MON08 16-Pin Header Signals tab - Status Area 336
MON08 Advanced Options Dialog Box 339
MON08 Cable Connecton Communications Type 332
MON08 Connection Manager Window - 16-Bit Header Signals Tab 335
MON08 Connection Manager Window - Advanced Settings tab 331
MON08 Connection Manager Window - Connect Target Tab 329
MON08 Connection View Register Files... Menu Option 341
MON08 Device Class Description 343
MON08 Expert Mode Programmer Window 340
MON08 Interface Connection 329
MON08 Interface debugger connection 329
MON08 Interface Selection Dialog Box 330
MON08 Power Cycle Control 333
MON08 Power Up/Power Down Controls 332
MON08 Tpd and Tpu Timing 331
Monitor Communication 562
MONITOR-HCS08 > Bus Trace 563
MONITOR-HCS08 > Erase Flash 562
MONITOR-HCS08 > Monitor Communications... 562
MONITOR-HCS08 > Select Derivative 563
MONITOR-HCS08 > Trigger Module Settings... 562
MONITOR-HCS08 > Vector Mirroring Setup... 562
MotoSIL 445, 490
Motosil 455, 499
MS 705
ms 107
Multilink Cyclone Pro - Debugging Limitations 377

Multilink/Cyclone Pro - Advanced Options Dialog Box 374
Multilink/Cyclone Pro - Advanced Settings Tab 365
Multilink/Cyclone Pro - Connection Manager Dialog Box 364
Multilink/Cyclone Pro - Connection Manager Window - Connect to Target Tab 362
Multilink/Cyclone Pro - Connection Manager Window -Advanced Settings Tab 365
Multilink/Cyclone Pro - Connection Manager Window Interface Details Area 369
Multilink/Cyclone Pro - Connection Manager Window STATUS Area 370
Multilink/Cyclone Pro - Connection Menu Added Options 374
Multilink/Cyclone Pro - Connection Menu Advanced Programming/Debug Option 374
Multilink/Cyclone Pro - Connection Menu View Register Files Option 376
Multilink/Cyclone Pro - Cycle Power Button 367
Multilink/Cyclone Pro - Device Class Description 379
Multilink/Cyclone Pro - Device Power Selection List 362
Multilink/Cyclone Pro - Expert mode Programmer WIndow 375
Multilink/Cyclone Pro - HC08 Device Extended Menus 373
Multilink/Cyclone Pro - Interface Selection Dialog Box 363
Multilink/Cyclone Pro - MON08 Cable Connection Communications Type 366
Multilink/Cyclone Pro - Power Up and Power Down Buttons 366
Multilink/Cyclone Pro - Remove a Manually Configured Interface 363
Multilink/Cyclone PRO - Set Connections Dialog Box 361
Multilink/Cyclone Pro - Start Expert Mode Programmer Option 375
Multilink/Cyclone Pro - Target MCU Security Bytes 368
Multilink/Cyclone Pro - Tpd And Tpu Timing Textboxes 365
Multilink/Cyclone Pro Connection Menu 372
Multilink/Cyclone Pro Connection Procedure 361
MultilinkCyclonePro Connection Menu - Device Option 372

N

NB 706
New 15
NOCR 708
-Nodefaults 10
NOLF 708
Nth Event 463

O

Object Info Bar 35
OBJPATH 793
Oct 69, 79, 104, 198
Octal 198
OPEN 709
Open Component 30
Open Configuration 15
Open Source File 115
Options 190
    Pointer As Array. 67
Options - Autosize 32
Options - Component Menu 32
OSC 767, 774
OSEK Kernel Awareness 180
OSEK ORTI 180, 181
OSEK RTK Inspector 182
OSPARAM.PRM 176
Outlinecolor 140
OUTPUT 710

P

P 711
P&E 567
Parameters 499
Paste 133
PATH 785

Pause 100
PAUSETEST 712
PEDebug 567
Percentage 57, 96
Periodical 67, 80
Personality file 451, 495
Pin Assignments, Connector 440, 487
Platform Board, Configuration 435, 483
Play 100
Pod 441
Pointer as Array 67, 71
Port to Display 135
Port Voltage Control Jumper Headers (J2–J4) 435, 484
Postload command file 29
postload.cmd 205
Preference panel 17, 18
Preferences dialog 16
Preload command file 29
preload.cmd 205
PRINTF 713
Procedure Chain 94
-Prod 10
Profiling 583, 604
Program flow rebuild gap 598
Project 783
PROJECT.INI 24, 445, 455, 490, 499, 782
project.ini 781
Properties 132
Protect DBG FIFO content from unexpected reads 596
PROTOCOL 767, 774
Protocol 450
PTRARRAY 713
PVCS 795

R

R command 252, 341, 356, 376, 393, 506, 544, 620
R/W Access 593
RAM 377
RD 714
Read 574, 593
Read Access 578
Read/Write 574
Read/Write Access 578
READACCESS 577
READWRITEACCESS 577
real time 5
Real Time Kernel Awarness 175
Real Time Kernels 175
RECORD 715
Record 100
Record continuously and DO NOT halt on trigger hit 591
Record continuously and halt on trigger hit 591
recording modes 426
Register 103
Register values 156, 157, 166
Registration 33
Relative Mode 143
Remove 132
Remove MON08 Interface connection 331
REPEAT 700, 715
Replay 101
requirements 403
RESET 716, 764, 771
Reset 454, 498
Reset command file 28
Reset signal connection 454, 498
Reset Target 25
reset.cmd 205
RESTART 716
Restart 23
RETURN 717
RHAPSODY 88
Right 133
ROM 377
RS 718
RS08 FCS - Configuration Procedure 607
RS08 FCS - Connection (HCS08FCS) Menu 608
RS08 FCS - Connection Menu Module Commands 609
RS08 FCS - Menu Device Option Exteded Menus 608
RS08 FCS - Peripheral Modules Commands 612
RS08 FCS - Set Connection Dialog Box 607
RS08 Full Chip Simulation 607

RS08 Multilink/Cyclone Pro - Additional Connection Menu Options 617
RS08 Multilink/Cyclone Pro - Advanced Programming/Debug Option 617
RS08 Multilink/Cyclone Pro - Connection Assistant Dialog Box 614
RS08 Multilink/Cyclone Pro - Connection Procedure 613
RS08 Multilink/Cyclone Pro - Device Option 616
RS08 Multilink/Cyclone Pro - Device Option Extended Menu 616
RS08 Multilink/Cyclone Pro - Expert Programmer Window 619
RS08 Multilink/Cyclone Pro - Set Connection Dialog Box 613
RS08 Multilink/Cyclone Pro - Set Connections Dialog Box 613
RS08 Multilink/Cyclone Pro - Start Expert Mode Programmer Option 618
RS08 Multilink/Cyclone Pro - View Register Files Option 620
RS08 MultilinkCyclonePro - Connection Menu 616
RS08 P&E Multilink/Cyclone PRO Connection 613
RTMEM 768
Run Menu 475
Run To Cursor 51, 115

S

S 719
SAVE 720
Save and Restore on load 573, 578
Save Configuration 15
Save Configuration As 16
Save Layout 132
SAVEBP 721
SC 756
Scope... 67
Scrolling 471
Scrolling Display 471
SDI 46
Search 472
    Event 472
    Frame 472
    Pattern 474
search order 796
Searching Order
    Assembly source files 796
    C source files 796
    Object files source files 796
select
    memory map 417
Select Derivative 563
Send to Back 133
Send to Front 133
Sequencer Setup 460
Sequential 462
Serial 440
Serial Monitor 588
SET 722
Set Breakpoint 51, 114
Set Connection Dialog Box - Full Chip Simulation Option 249
Set DBGCA 575
Set DBGCB 575
Set Target 30
Set Trigger A 581
Set Trigger Address 574
Set Trigger Address B 575
Set Trigger B 581
Set TriggerAddress 569
Set Zero Base 108
SETCOLORS 722
Setup 132
setup 407
Show Breakpoints 51, 115
Show Location 51, 115, 594, 598
Show Markpoints... 571
Show Protocol 450
ShowLocation 470
SHOWPROT 456, 500
SIG 763, 770
Signals 453, 498
Simulation 5
Single Step 23
Size 133
Size of Port 135

SLAY 723
SLINE 723
Sloping 140
Small Borders. 19
SMEM 724
SMOD 725
SofTec HC08 - CodeWarrior and SofTec HC08 Connection 395
SofTec HC08 - Communication Settings Dialog Box 400
SofTec HC08 - First Steps Using the Stationery Wizard 236
SofTec HC08 - From Within an Existing Project 397
SofTec HC08 - inDart-HC08 Connection Menu Options 398
SofTec HC08 - MCU Configuration Dialog Box 398
SofTec HC08 - Set Connection Dialog Box 397
SofTec HC08 - Technical Considerations 234, 395
SofTec HC08 - Using the Stationery Wizard 396
SofTec HC08 - Wizard Connection Selection 396
Softec HC08 Connection 395
SofTec HCS08 - CodeWarrior and SofTec HCS08 Connection 547
SofTec HCS08 - Communication Settings Dialog Box 553
SofTec HCS08 - Connection Menu Options 551
SofTec HCS08 - First Steps From Within an Existing Project 550
SofTec HCS08 - First Steps Using the Stationery Wizard 548
SofTec HCS08 - MCU Configuration Dialog Box 551, 552
SofTec HCS08 - Set Connection Dialog Box - 550
SofTec HCS08 - Technical Considerations 547
SofTec HCS08 - Wizard Connection Selection 548
Softec HCS08 Connection 547
SofTec RS08 - CodeWarrior and SofTec RS08 Connection 623
SofTec RS08 - Communication Settings Dialog Box 628
SofTec RS08 - Connection Menu Options 627
SofTec RS08 - First Steps From Within an Existing Project 626
SofTec RS08 - First Steps Using the Stationery Wizard 624
SofTec RS08 - MCU Configuration Dialog Box 627, 628
SofTec RS08 - Set Connection Dialog Box 626
SofTec RS08 - Technical Considerations 623
SofTec RS08 - Wizard Connection Selection 624
Softec RS08 Connection 623
SofTec-HCS08 > About 552
SofTec-HCS08 > Bus Trace 552
SofTec-HCS08 > MCU Configuration 552
SofTec-HCS08 > Trigger Module Settings... 552
SofTec-RS08 > About 628
SofTec-RS08 > MCU Configuration 628
SP 757
SPC 726
Splitting View 58
SPROC 727
SQ 760
SREC 728
ST 747
ST.SectionTitle 1, 3, 231, 395, 403, 431, 481, 501, 547, 555, 605, 631, 633, 745, 775, 777, 799, 801, 803
ST1619-HDS
  Postload command file 29
  Preload command file 29
  Reset command file 28
  Startup command file 28
Start 101
Start recording after trigger hit and halt when the fifo is full 591, 592
Start/Continue 23
Starting an Application 194
startup 781
Startup command file 28
startup.cmd 205
Statistics 97
Status Bar 13, 19

Status bar 579
Status register bits 103
Step In 195
    Assembly Instruction 197
    Source Instruction 195
Step Out 23, 195
    Function Call 196
Step Over 23, 195, 196
STEPINTO 729
STEPOUT 730
STEPOVER 731
STOP 732
Stopping an Application 194
Symbolic 68, 69, 198
System
    Components 432, 482
    Connections 485
        437
system components 405
system features 404
system requirements 403

T

-T 9
T 733
-Target 10
Target 445, 490
target cable 406
target head adapter
    as additional component 406
task 175
TD 750
TE 751
term 424
TESTBOX 734
Text 139
Text Mode 142
Textcolor 142
Textual 599
Textual Display 466
Textual display 599
Tile 32
time tag 428
Time Tag Clock 463
Timer Update 59
TMP 794
Toolbar 12, 19
    Customizing 19
ToolTips 116
ToolTips Activation 110
ToolTips format 110
ToolTips mode 110
Top 133
trace
    arming 428
    time tag 428
    viewing data 429
Trace Buffer 457
Trace component 597
Trace Modes 456
Trace window 597
traced 598
Trigger 466
Trigger A 569, 571
Trigger address edition 593
Trigger B 569, 571
trigger B 586
Trigger Events 458
Trigger Module Settings 552, 568
Trigger Module Settings... 562
Trigger Setup 458
True Time Simulator and Real Time Debugger
    Configuration 190
    Default Layout Configuration 782
    Demo Version Limitations 6
    Drag and Drop 38
    Engine 5
    Layout 783
    Project 783
    project.ini 782
    Running from a command line 9
    Smart User Interface 36
    Tool tip 13
    User Interface 36
    Using on Windows 95 or Windows NT 4.0/
        WIN2000 190
TT 761
TUPDATE 734

Type 593

U

UDec 69, 80, 104, 198
UNDEF 735
undefined 594
UNFOLD 737
Unfold 120
Unfold All Text 120
Unfolding 112
    Mark 112
Unsigned Decimal 198
UNTIL 738
UPDATERATE 738
USELIBPATH 795
User 68
User's Manual 399

V

VA 744, 762
Variable
    Address 200
    Displaying Global Variables 197
    Displaying Local Variables 197
    Editing Value 199
    Format 64
    Local and Global 64
    Mode 67
    Scope 64
    Showing Location 200
    Type 64
    Value 198
Vector Mirroring Setup... 562
VER 739
Version number 33
Vert. Text Alignment 142
Vertical Size 133
Viewing Data 466
viewing data 429
VisualizationTool
    7 Segment Display 139
    Analog 135
    Bar 136
    Bitmap 137, 138
    Demo 144
    Demo limitation 144
    Demo Version Limitations 144
    DILSwitch 138
    Instrument 134
    Knob 138
    LED 139
    Setup 134
    Switch 140
    Text 141

W

-W 10
WAIT 740
Watchpoint
    Checking condition 162
    Command 167
    Conditional 166, 174
    Counting 165, 174
    Definition 145
    Deleting 167, 173
    Read 163, 172
    Read, Write 146
    Read/Write 164, 173
    Write 164
watchpoint 568, 574
Watchpoints 456
WB 741
WHILE 700, 742
Width 135
Windows 779
WinEdit 779
WL 743
Word 79
Word size 78
WorkDir 190
WorkingDirectory 190
Write 574, 593
Write Access 578
WRITEACCESS 577
WW 743

X

X-Position 134

Y
Y-Position 134

Z
ZOOM 744
Zoom 469
Zoom in 67